Productivity Engineering in the UNIX† Environment

The PPP Simulator: User's Manual and Report

Technical Report

S. L. Graham
Principal Investigator

(415) 642-2059

"The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government."

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1987

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION unclassified	1b. RESTRICTIVE MARKINGS AD-A179 495
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION The Regents of the University of California 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION SPAWAR
6c. ADDRESS *(City, State, and ZIP Code)* Berkeley, California 94720	7b. ADDRESS *(City, State, and ZIP Code)* Space and Naval Warfare Systems Command Washington, DC 20363-5100
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*

* The PPP Simularor: User's Manual and Report

12. PERSONAL AUTHOR(S)

* Barry Fagin

13a. TYPE OF REPORT technical	13b. TIME COVERED FROM ______ TO ______	14. DATE OF REPORT *(Year, Month, Day)* * November, 1986	15. PAGE COUNT * 151

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Enclosed in paper.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE *(Include Area Code)* / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

(1)

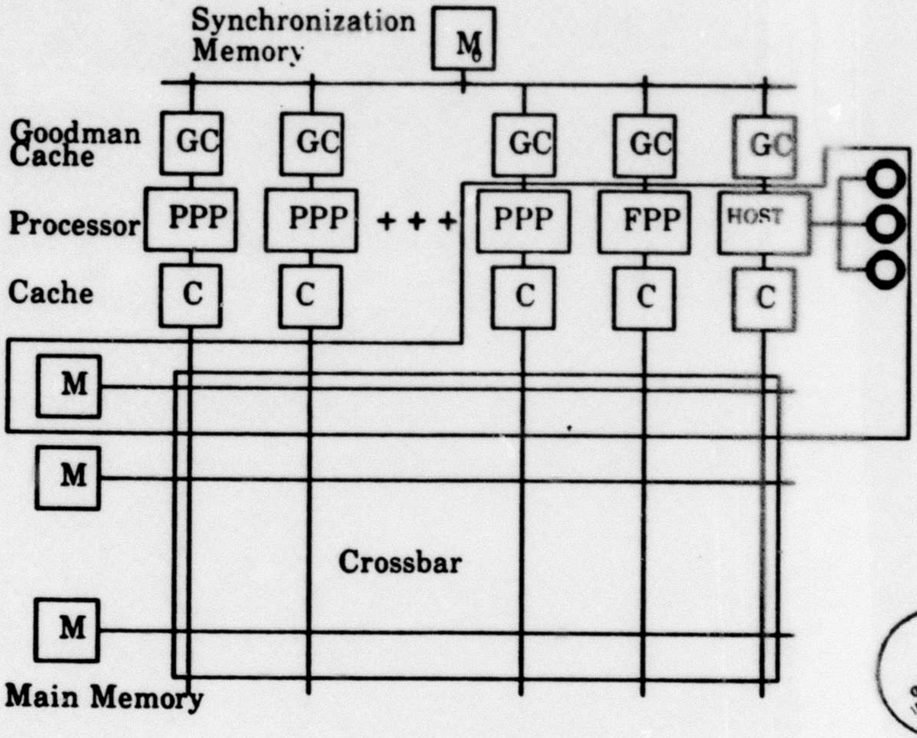
The PPP Simulator: User's Manual and Report

Barry Fagin (fagin@ji.berkeley.edu)
Computer Science Division
570 Evans Hall
University of California
Berkeley, California 94720

1. Introduction

This paper is the user's manual for the PPP simulator. The PPP simulator is an extension of Tep Dobry's PLM simulator; thus the reader is assumed to be familiar with the PLM simulator user's manual.

The PPP simulator provides an approximate modeling of the Aquarius multiprocessor system, shown below: see next page



The Aquarius System

(The area inside the dotted line represents the system modeled by the PLM simulator). The memory systems are not simulated in detail: a probabalistic model is used to estimate their effect on execution time.

Using the simulator, a user can run programs to completion, run individual processes one instruction at a time, examine the current state of any process, and obtain performance measurements. The simulator is still under development; changes will be reflected in later versions of this manual.

2. The PPP Execution Model

The main difference between the PPP and the PLM is the difference between their underlying execution models: The PLM uses a sequential execution model, while the PPP uses a parallel execution model This parallelism is achieved using *processes*, communicating via *messages*.

2.1. Processes in the PPP

A process in the PPP is essentially a virtual PLM; each process has its own copy of the PLM register set, along with its own Heap, Trail, Stack, and PDL. Processes are created by special W-instructions. There are only two kinds of processes: AND-processes, and OR-processes. These are the means by which AND-parallelism and OR-parallelism are achieved.

A process is an AND-process if it must succeed in order for its parent to succeed Instructions that create AND-processes are only generated by the compiler if the created process is guaranteed never to bind shared variables. For example, in the Prolog clause:

```
foo :- a(X), b(Y).
```

and AND-process would be created for the goal "a(X)". In this way, the PPP supports AND-parallelism.

A process is an OR-process if it or one of its siblings must succeed in order for their parent to succeed. OR-processes, unlike AND-processes, always bind shared variables. Thus OR-processes use a special dereferencing and binding mechanism when examining variables passed to them from their parent In the Prolog clause:

```
foo :- a(X).
a(1).
a(2).
a(3).
a(4).
```

OR-processes could be created for all the unit clauses in the procedure for a. In this way, the PPP supports OR-parallelism.

A process may be in one of three states: running (currently executing on a processor), runnable (ready to execute as soon as a processor becomes available) and sleeping (waiting for a message from another process).

2.2. Messages in the PPP

There are four main kinds of messages in the PPP: SUCCESS, FAIL, NEXT_ANSWER, and KILL. SUCCESS and FAIL messages are only sent from child processes to parents; they report the success or failure of the sending process in solving its goal. NEXT_ANSWER and KILL messages are only sent from parent processes to children; they induce backtracking upon the receiving process, or terminate execution

An additional message, CUT, is used to implement the cut operator in an OR-parallel procedure.

The receipt of a message by a process may cause any number of things to happen. A synchronization counter may be decremented, backtracking may occur, a process's state may be changed from sleeping to runnable, and so forth. The type of action performed depends on the types of the processes and the message being sent.

3. The Role of the Host in the PPP

The Host processor has a much larger role in the PPP in in the PLM. In addition to handling external builtins, it is now responsible for process creation, scheduling, and interprocess communication. The host can be interrupted by any of the processors and can also interrupt them. Extending the notion of the communication area of the PLM, each processor has its own communication area which the processor can read and write. This is used for host-processor communication.

To assist in the task of process management, the host maintains a *process table*, a global data structure containing the state of all processes. When an instruction is executed by a processor that will create a process, the processor communicates the relevant information to the host and continues execution. The host then constructs the state of the new process and attempts to find an available processor. If one is available, the new process is loaded on the processor and begins execution. Otherwise, the process is marked 'runnable' and kept in the process table, waiting for a processor to be made available.

The process table is also used in interprocess communication. When a processor wishes to send a message to another process, it supplies the message and the process id of the destination to the host. Since the host has access to all processes via the process table, it can immediately take the appropriate action, even if the receiving process is sleeping.

4. Using the Simulator

To invoke the simulator, type

```
ppp [options] file
```

The file is expected to contain W-code corresponding to a Prolog program; normally, this file is produced by a Prolog compiler. At the moment, however, only the PLM sequential compiler is available. Thus concurrent programs must be hand-compiled.

The name of the file must end in '.w'.

5. Options

Several options may be passed to the simulator:

- -d Go immediately into debug mode. The simulator will read in the program, initialize the system, and then prompt for debugging commands. Using the simulator in debug mode is described in the following section.
- -i Produce instrumentation information. If the file being simulated is 'foo.w', this option will produce the files 'foo.data' and 'foo.ggraph'. The .data file contains various performance measurements, while the .ggraph file may be used as input to ggraph to produce a compressed graph of runnable processes versus time.
- -l Print out the label table before simulation.
- -np Simulate a p-processor system (the default is p=16). Note that "-n1" will give sequential operation, similar to the PLM simulator.

-c Print out the code space before simulation.

-p Print out the procedure table before simulation.

-s Check for system saturation when simulating. When this option is specified, instructions that normally create processes will not do so if no processors are available. The default is to create processes whenever the appropriate instructions are executed.

6. Debug Mode

Debug mode is used for stepping through programs and examining the state of the system. When executing in debug mode, the simulator will print out messages reporting changes of process status, creation of new processes, and so forth, Debug mode is indicated by the prompt "dbg>". The following commands may be used in debug mode. All arguments are in hex. These commands supersede those listed in the PLM simulator user's manual; some commands listed there are no longer supported or have changed

b addr[,pid]
: Set the breakpoint at 'addr'. A process id may also be specified. Currently, the simulator supports one breakpoint.

c Continue execution.

cp [addr]
: Print the choice point at 'addr'. If 'addr' is not specified, the value is taken from the B register. Thus if 'addr' is specified, it should be one greater than the last item in the choice point to be printed.

e [addr]
: Print the environment at 'addr' If 'addr' is not specified, the value is taken from the E register.

ex [addr]
: Print the extended environment at 'addr'. If 'addr' is not specified, the value is taken from the E register.

p id Print the state of process #id

pc a,b
: Print the contents of the code space from 'a' to 'b'.

pd a,b
: Print the contents of the data space from 'a' to 'b'.

pl Print the label table

pp Print the procedure table.

pr Print the current values of all registers.

ps Print the system status. This shows all processors with running or sleeping processes and their corresponding pid's.

q Quit.

s Single step. Find the next processor with a running process and simulate it for one instruction.

t id Trace execution of the indicated process. When the process executes an instruction, the simulator suspends and enters debug mode. Execution may be resumed with 'c' or 's'. Currently up to eight processes may be traced.

u pid Untrace process #pid.

w val[,pid]
: Write the term represented by 'val' on the console This is essentially a call to the

Prolog 'write' function. This command should be used carefully, as an improper value for 'val' can cause the simulator to crash. If a process id is specified, the value will be written as seen by the specified process. This may be different if, for example, the process is an OR process examining a locally bound variable.

wi pid
Print the binding window for the indicated process.

If, instead of a command, a carriage return is used, the command defaults to the last command entered. Thus 's' followed by a series of returns will step through the system.

Upon query success or goal failure, the simulator will respond with the prompt "quit>". A carriage return here will exit the program, while typing 'd' will return the simulator to debug mode, where the previous commands may be used.

7. The .data and .ggraph Files

If the '-i' option is used, the simulator will produce a .data file containing performance measurements. This file contains numerous instrumentation information, including:

(1) All instrumentation information provided by the PLM simulator.

(2) Information indicating the number of processors simulated, and whether the program simulated employed AND-parallelism, OR-parallelism, and/or intelligent backtracking.

(3) The number of context swaps.

(4) The number of microcycles executed by each processor (denoted by uCYCLES). Note that this is not a direct measure of execution time, because during execution processors may wait for access to the host or a synchronization lock. Microcycles that read or write may also take longer.

(5) The number of reads and writes performed by each processor (denoted by RDS and WTS).

(6) The number of requests made of the host by each processor (denoted by EF, for "external functions"). This includes requests for interprocess communication, process creation, termination, etc.

(7) The number of critical sections entered by each processor (denoted by CSEC).

(8) An estimate of the execution time and the assumptions on which the estimate is based. The execution time for processor p is given by

$$T_p = uCYCLES_p - RDS_p - WTS_p + RDS_p * T_r + WTS_p * T_w + CSEC * T_s + EF * T_{ef}$$

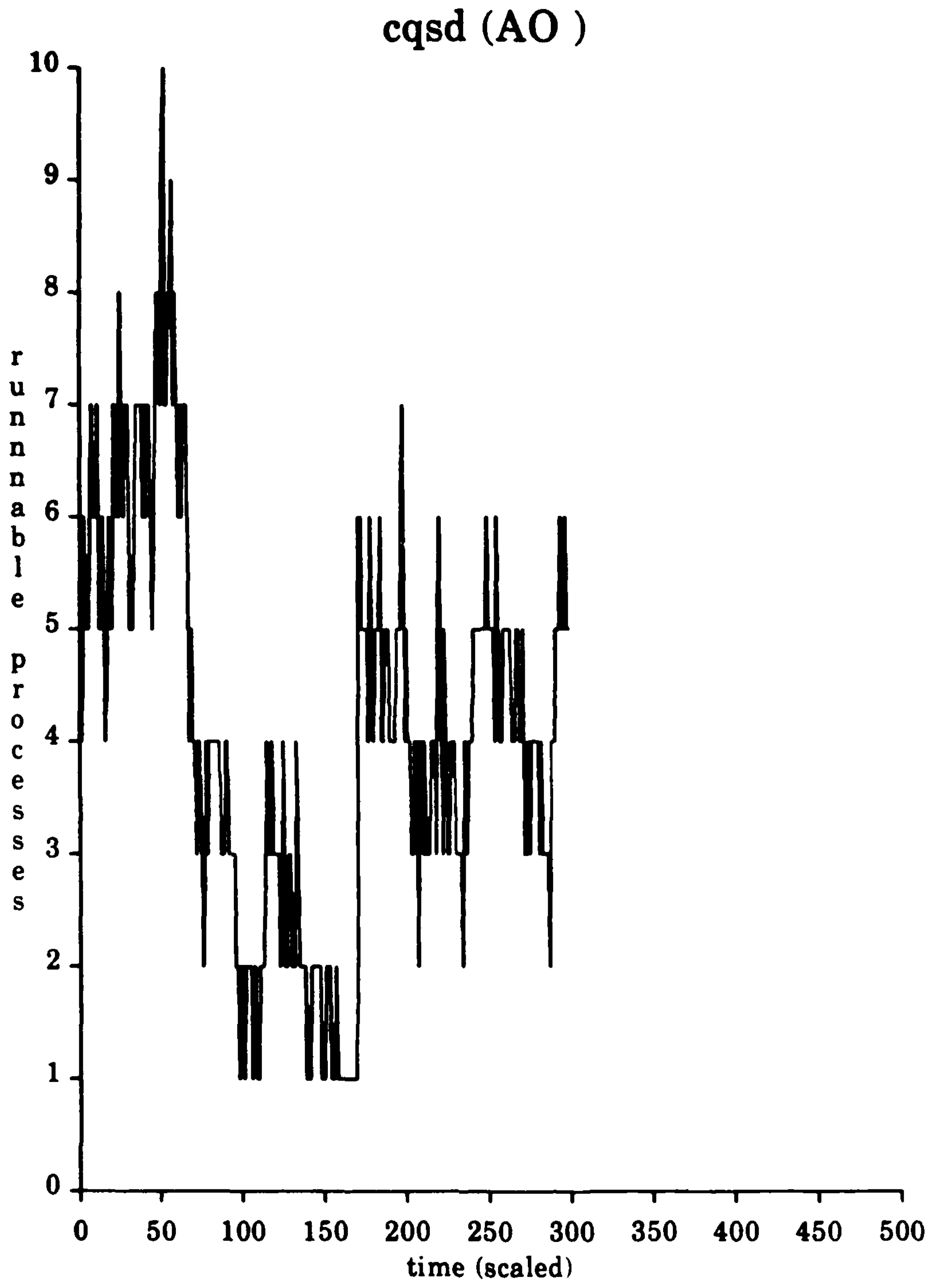
where T_r is the time for a read, T_u the time for a write, T_s the time to get a synchronization lock, and T_{ef} the time it takes for the host to respond to a request from a processor. The execution time for the program is that of the processor with the longest execution time.

(9) The "execution time weights". A processor may spend its time executing internally, reading, writing, accessing the host and waiting for a response, or accessing a synchronization variable. These weights give the ratio of the number of times each of these operations occurred to their sum, to enable the user to estimate the effect of errors in the various parameters on the total execution time.

(10) The number of access made to the synchronization memory and the ratio of these to the total number of memory accesses, expressed as a percent.

(11) The average number of runnable and running processes.

If the '-i' option is indicated the simulator will also produce a .ggraph file. This can be used as an input to ggraph to produce an approximate graph of runnable processes versus time. Each time a process executes one instruction, the number of runnable and running processes is recorded in a temporary file. At program termination, this data in this file is compressed into slightly less than 500 data points, since 500 points is the maximum number of points ggraph can process. Typing "ggraph foo.ggraph" will produce the file foo.grn, which can then be printed using gprint or grn and ditroff. The graph will contain the prefix of the benchmark's filename, and a character string containing all or part of the letters "AOI", indicating whether AND-parallelism, OR-parallelism, and/or intelligent backtracking was used by the program. The following is a typical process graph:



8. Compatibility with PLM

The PPP simulator can run all PLM programs. It uses the same data types, and provides all the builtin functions with the exception of 'assert' and 'retract'. Its instruction set includes that of the PLM. The only incompatibility between the two simulators is the syntax of some of the debugging commands: some old PLM commands are no longer supported

9. References

[Dob84] T. P. Dobry, "PLM Simulator Reference Manual", *Working Paper 3.3*, Berkeley, CA, Sep. 84.

[DCD85] T. P. Dobry, J. H. Chang, A. M. Despain and Y. N. Patt, "Extending a Prolog Machine for Parallel Execution", *Proceedings of Hawaii Int. Conf. on System Science 86*, 1985.

[FaD85] B. Fagin and T. Dobry, *The Berkeley PLM Instruction Set: An Instruction Set for Prolog*, Computer Science Division, University of California, Berkeley, Sep. 1985. Research Report No. UCB/Computer Science Dpt. 86/257.

[War83] D. H. D. Warren, *An Abstract Prolog Instruction Set*, Computer Science and Technology Division, SRI, Menlo Park, CA, Oct. 1983. Technical Note 309, Artificial Intelligence Center.

10. Appendix: Sample Program, Compilation, and Simulation

The following Prolog program:

```
main :- concat(X,Y,[a,b,c,d,e]),write(X),nl,write(Y),nl,nl,fail.

concat([],L,L).
concat([A|B],C,[A|D]) :- concat(B,C,D).
```

Compiles into:

```
procedure main/0

_332:
        put_list  X3
        unify_constant  a
        unify_constant  b
        unify_constant  c
        unify_constant  d
        unify_constant  e
        unify_nil
        allocate
        put_variable  Y2,X1
        put_variable  Y1,X2
        call  concat/3,2
        put_unsafe_value  Y2,X1
        escape  write/1
        escape  nl/0
        put_unsafe_value  Y1,X1
        escape  write/1
        escape  nl/0
          escape  nl/0
        fail

  procedure concat/3
          switch_on_term C1,C2,fail

  C1a:    try_me_else_p 2,C2a
  C1:     get_nil A1
          get_value      A2,A3
          proceed

  C2a:    trust_me_else_p        2
  C2:     get_list A1
          unify_variable X4
          unify_cdr      A1
          get_list A3
          unify_value    X4
          unify_cdr      A3
          execute        concat/3
  end
```

Sample execution run on simulator:

```
Script started on Thu Aug 28 09:16:15 1986
1 [ji] -> ppp -d Benchmarks/Par/ccon6.w
pau load  /a/hprg/fagin/PPP1/Benchmarks/Par/concatOP.w
Cspace = 31
pau load  Benchmarks/Par/ccon6.w
Cspace = 31
dbg> s
Running process 0
CLOC 00000000: put_list        X3
dbg>
Running process 0
CLOC 00000001: unify_constant        a
dbg>
Running process 0
CLOC 00000002: unify_constant        b
dbg>
Running process 0
CLOC 00000003: unify_constant        c
dbg>
Running process 0
CLOC 00000004: unify_constant        d
dbg>
Running process 0
CLOC 00000005: unify_constant        e
dbg>
Running process 0
CLOC 00000006: unify_nil
dbg>
Running process 0
CLOC 00000007: allocate
dbg>
Running process 0
CLOC 00000008: put_variable Y2,X1
dbg>
Running process 0
CLOC 00000009: put_variable Y1,X2
dbg>
Running process 0
CLOC 0000000a: call   concat/3,2
dbg>
Running process 0
CLOC 00000013: switch_on_term        C1,C2,fail
dbg>
Running process 0
CLOC 00000014: try_me_else_p         2,C2a
dbg>
Running process 0
CLOC 00000018: trust_me_else_p     2
Created OR process, parent = 0, id = 1
dbg> ps
0       0       RUNNING
```

```
1      1       RUNNING
dbg> s
Running process 1
CLOC 00000019: get_list        A1
dbg>
Running process 0
CLOC 00000015: get_nil         A1
dbg>
Running process 1
CLOC 0000001a: unify_variable        X4
dbg>
Running process 0
CLOC 00000016: get_value    A2,A3
dbg>
Running process 1
CLOC 0000001b: unify_cdr    A1
dbg>
Running process 0
CLOC 00000017: proceed
dbg>
Running process 1
CLOC 0000001c: get_list        A3
dbg>
Running process 0
CLOC 0000000b: put_unsafe_value    Y2,X1
dbg>
Running process 1
CLOC 0000001d: unify_value  X4
dbg>
Running process 0
CLOC 0000000c: escape          write/1
 dbg>
Running process 1
CLOC 0000001e: unify_cdr    A3
dbg>
Running process 0
CLOC 0000000d: escape          nl/0

dbg>
Running process 1
CLOC 0000001f: execute         concat/3
dbg>
Running process 0
CLOC 0000000e: put_unsafe_value    Y1,X1
dbg>
Running process 1
CLOC 00000013: switch_on_term      C1,C2,fail
dbg>
Running process 0
CLOC 0000000f: escape          write/1
a b c d e dbg>
Running process 1
CLOC 00000014: try_me_else_p       2,C2a
```

```
dbg>
Running process 0
CLOC 00000010: escape        nl/0

dbg>
Running process 1
CLOC 00000018: trust_me_else_p     2
Created OR process, parent = 1, id = 2
dbg> ps
0     0      RUNNING
1     1      RUNNING
2     2      RUNNING
dbg> p 0
PROCESS 0
parent = ffffff; kind = AND; c0 = FORWARD; c1 = ffffff; state = RUNNING
AX[1] = 00000080;     00000080;     00000080;     9ffffff;
AX[5] = 9ffffff;        9ffffff; 9ffffff; 9ffffff;
P = 00000011; CP = 0000000b;     E = 00004000;        B = 0000401a
TR = 00007c02;        H = 00000086;       HB = 00000086;       S = 00000080
N = 00000002; W = 00000000; PDL = 00008000; E_FF = 00000000
cut = 00000000; ictr = 00000007; parB = 00000000; i = 00000000
Wbase = 00000000; HPbase = 00000080; STKbase = 00004000; TRLbase = 00007c00
dbg> p 1
PROCESS 1
parent = 00000000; kind = OR; c0 = FORWARD; c1 = ffffff; state = RUNNING
AX[1] = 80008041;     80004004;     00000081;     80008040;
AX[5] = 9ffffff;        9ffffff; 9ffffff; 9ffffff;
P = 00000015; CP = 00000000;     E = 000083ff; B = 00008414
TR = 00008800;        H = 00008042;       HB = 00008042;       S = 00000081
N = 00000000; W = 00008002; PDL = 00008880; E_FF = 00000000
cut = 20000000; ictr = 0000000a; parB = 0000401a; i = 00000002
Wbase = 00008000; HPbase = 00008040; STKbase = 00008400; TRLbase = 00008800
dbg> p 2
PROCESS 2
parent = 00000001; kind = OR; c0 = FORWARD; c1 = ffffff; state = RUNNING
AX[1] = 80008041;     80004004;     00000081;     80008040;
AX[5] = 9ffffff;        9ffffff; 9ffffff; 9ffffff;
P = 00000019; CP = 00000000;     E = 00008c7f; B = 00008c80
TR = 00009080;        H = 000088c0;       HB = 000088c0;       S = 000088c0
N = 00000000; W = 00008880; PDL = 00009100; E_FF = 00000000
cut = 00000000; ictr = 00000000; parB = 00008414; i = 00000002
Wbase = 00008880; HPbase = 000088c0; STKbase = 00008c80; TRLbase = 00009080
dbg> s
Running process 2
CLOC 00000019: get_list      A1
dbg>
Running process 0
CLOC 00000011: escape        nl/0

dbg>
Running process 1
CLOC 00000015: get_nil       A1
dbg>
```

```
Running process 2
CLOC 0000001a: unify_variable        X4
dbg>
Running process 0
CLOC 00000012: fail
  backtracking...
  restoring from OR choice point
  now listening to p1
  putting p0 to sleep
dbg>
Running process 1
CLOC 00000016: get_value     A2,A3
dbg>
Running process 2
CLOC 0000001b: unify_cdr     A1
dbg>
Running process 1
CLOC 00000017: proceed
  p1 sending SUC to p0
  putting p1 to sleep
dbg>
Running process 2
CLOC 0000001c: get_list       A3
dbg>
Running process 0
CLOC 0000000b: put_unsafe_value     Y2,X1
dbg>
Running process 2
CLOC 0000001d: unify_value  X4
dbg>
Running process 0
CLOC 0000000c: escape         write/1
a dbg>
Running process 2
CLOC 0000001e: unify_cdr     A3
dbg>
Running process 0
CLOC 0000000d: escape         nl/0

dbg>
Running process 2
CLOC 0000001f: execute        concat/3
dbg>
Running process 0
CLOC 0000000e: put_unsafe_value     Y1,X1
dbg>
Running process 2
CLOC 00000013: switch_on_term      C1,C2,fail
dbg>
Running process 0
CLOC 0000000f: escape         write/1
b c d e dbg>
Running process 2
```

```
CLOC 00000014: try_me_else_p        2,C2a
dbg>
Running process 0
CLOC 00000010: escape        nl/0

dbg>
Running process 2
CLOC 00000018: trust_me_else_p      2
Created OR process, parent = 2, id = 3
dbg>
Running process 3
CLOC 00000019: get_list        A1
dbg>
Running process 0
CLOC 00000011: escape        nl/0

dbg>
Running process 2
CLOC 00000015: get_nil        A1
dbg>
Running process 3
CLOC 0000001a: unify_variable        X4
dbg>
Running process 0
CLOC 00000012: fail
  backtracking...
  restoring from OR choice point
  p0 sending NA to p1
  putting p0 to sleep
dbg>
Running process 1 BACKWARD
  backtracking...
  restoring from OR choice point
  now listening to p2
  putting p1 to sleep
dbg> ps
0       0       SLEEPING
1       1       SLEEPING
2       2       RUNNING
3       3       RUNNING
dbg>
0       0       SLEEPING
1       1       SLEEPING
2       2       RUNNING
3       3       RUNNING
dbg>
0       0       SLEEPING
1       1       SLEEPING
2       2       RUNNING
3       3       RUNNING
dbg> s
Running process 2
CLOC 00000016: get_value    A2,A3
```

```
dbg>
Running process 3
CLOC 0000001b: unify_cdr     A1
dbg>
Running process 2
CLOC 00000017: proceed
  p2 sending SUC to p1
  putting p2 to sleep
dbg> c
a b
c d e

a b c
d e

a b c d
e

a b c d e


goal failure
goal failure
Stopped.
cpu time is 3.840000 sec
2 [ji] -> ^D
script done on Thu Aug 28 09:18:16 1986
```

Productivity Engineering in the UNIX† Environment

The PPP Simulator: User's Manual and Report

Technical Report

S. L. Graham
Principal Investigator

(415) 642-2059

Contract No. N00039-84-C-0089

August 7, 1984 - August 6, 1987

Arpa Order No. 4871

†UNIX is a trademark of AT&T Bell Laboratories

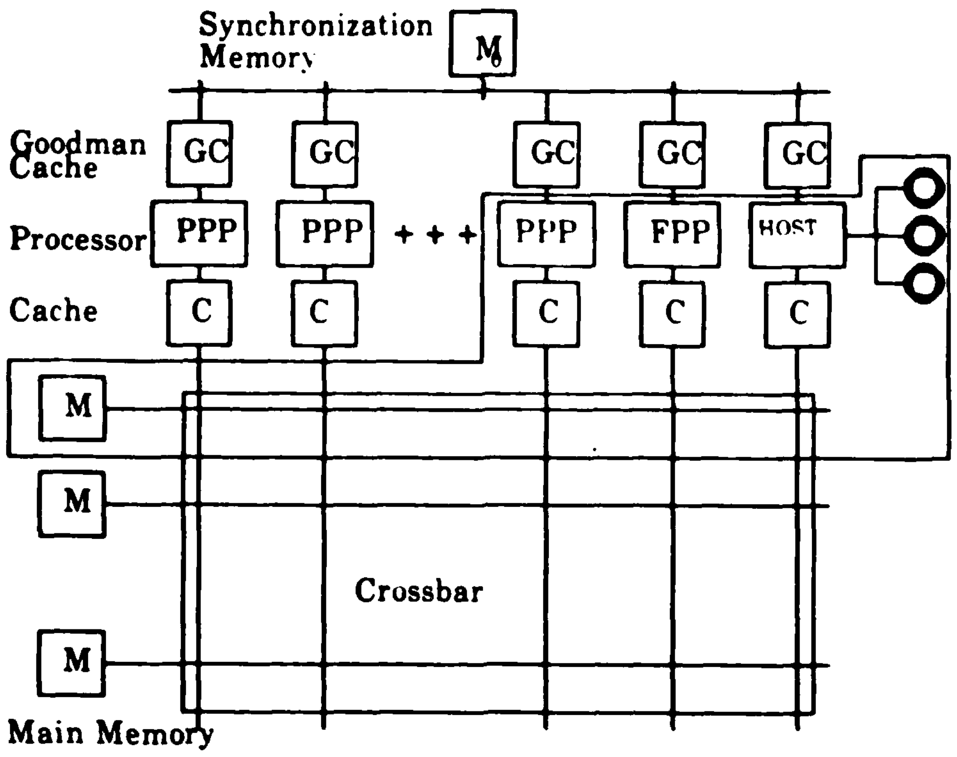
The PPP Simulator: User's Manual and Report

Barry Fagin (fagin@ji.berkeley.edu)
Computer Science Division
570 Evans Hall
University of California
Berkeley, California 94720

1. Introduction

This paper is the user's manual for the PPP simulator. The PPP simulator is an extension of Tep Dobry's PLM simulator; thus the reader is assumed to be familiar with the PLM simulator user's manual.

The PPP simulator provides an approximate modeling of the Aquarius multiprocessor system, shown below:



The Aquarius System

(The area inside the dotted line represents the system modeled by the PLM simulator. The memory systems are not simulated in detail: a probabalistic model is used to estimate their effect on execution time.

Using the simulator, a user can run programs to completion, run individual processes one instruction at a time, examine the current state of any process, and obtain performance measurements. The simulator is still under development; changes will be reflected in later versions of this manual.

2. The PPP Execution Model

The main difference between the PPP and the PLM is the difference between their underlying execution models: The PLM uses a sequential execution model, while the PPP uses a parallel execution model. This parallelism is achieved using *processes*, communicating via *messages*.

2.1. Processes in the PPP

A process in the PPP is essentially a virtual PLM; each process has its own copy of the PLM register set, along with its own Heap, Trail, Stack, and PDL. Processes are created by special W-instructions. There are only two kinds of processes: AND-processes, and OR-processes. These are the means by which AND-parallelism and OR-parallelism are achieved.

A process is an AND-process if it must succeed in order for its parent to succeed. Instructions that create AND-processes are only generated by the compiler if the created process is guaranteed never to bind shared variables. For example, in the Prolog clause:

```
foo :- a(X), b(Y).
```

and AND-process would be created for the goal "a(X)". In this way, the PPP supports AND-parallelism.

A process is an OR-process if it or one of its siblings must succeed in order for their parent to succeed. OR-processes, unlike AND-processes, always bind shared variables. Thus OR-processes use a special dereferencing and binding mechanism when examining variables passed to them from their parent. In the Prolog clause:

```
foo :- a(X).
a(1).
a(2).
a(3).
a(4).
```

OR-processes could be created for all the unit clauses in the procedure for a. In this way, the PPP supports OR-parallelism.

A process may be in one of three states: running (currently executing on a processor), runnable (ready to execute as soon as a processor becomes available) and sleeping (waiting for a message from another process).

2.2. Messages in the PPP

There are four main kinds of messages in the PPP: SUCCESS, FAIL, NEXT_ANSWER, and KILL. SUCCESS and FAIL messages are only sent from child processes to parents; they report the success or failure of the sending process in solving its goal. NEXT_ANSWER and KILL messages are only sent from parent processes to children; they induce backtracking upon the receiving process, or terminate execution.

An additional message, CUT, is used to implement the cut operator in an OR-parallel procedure.

The receipt of a message by a process may cause any number of things to happen. A synchronization counter may be decremented, backtracking may occur, a process's state may be changed from sleeping to runnable, and so forth. The type of action performed depends on the types of the processes and the message being sent.

3. The Role of the Host in the PPP

The Host processor has a much larger role in the PPP in in the PLM. In addition to handling external builtins, it is now responsible for process creation, scheduling, and interprocess communication. The host can be interrupted by any of the processors and can also interrupt them. Extending the notion of the communication area of the PLM, each processor has its own communication area which the processor can read and write. This is used for host-processor communication.

To assist in the task of process management, the host maintains a *process table*, a global data structure containing the state of all processes. When an instruction is executed by a processor that will create a process, the processor communicates the relevant information to the host and continues execution. The host then constructs the state of the new process and attempts to find an available processor. If one is available, the new process is loaded on the processor and begins execution. Otherwise, the process is marked 'runnable' and kept in the process table, waiting for a processor to be made available

The process table is also used in interprocess communication. When a processor wishes to send a message to another process, it supplies the message and the process id of the destination to the host. Since the host has access to all processes via the process table, it can immediately take the appropriate action, even if the receiving process is sleeping

4. Using the Simulator

To invoke the simulator, type

```
ppp [options] file
```

The file is expected to contain W-code corresponding to a Prolog program, normally, this file is produced by a Prolog compiler. At the moment, however, only the PLM sequential compiler is available. Thus concurrent programs must be hand-compiled

The name of the file must end in '.w'.

5. Options

Several options may be passed to the simulator:

-d Go immediately into debug mode. The simulator will read in the program, initialize the system, and then prompt for debugging commands. Using the simulator in debug mode is described in the following section.

-i Produce instrumentation information. If the file being simulated is 'foo.w', this option will produce the files 'foo.data' and 'foo.ggraph'. The .data file contains various performance measurements, while the .ggraph file may be used as input to ggraph to produce a compressed graph of runnable processes versus time.

-l Print out the label table before simulation.

-np Simulate a p-processor system (the default is p=16). Note that "-n1" will give sequential operation, similar to the PLM simulator.

-c Print out the code space before simulation.

-p Print out the procedure table before simulation.

-s Check for system saturation when simulating. When this option is specified, instructions that normally create processes will not do so if no processors are available. The default is to create processes whenever the appropriate instructions are executed.

6. Debug Mode

Debug mode is used for stepping through programs and examining the state of the system. When executing in debug mode, the simulator will print out messages reporting changes of process status, creation of new processes, and so forth, Debug mode is indicated by the prompt "dbg>". The following commands may be used in debug mode. All arguments are in hex. These commands supersede those listed in the PLM simulator user's manual; some commands listed there are no longer supported or have changed

b addr[,pid]
: Set the breakpoint at 'addr'. A process id may also be specified. Currently, the simulator supports one breakpoint.

c Continue execution.

cp [addr]
: Print the choice point at 'addr'. If 'addr' is not specified, the value is taken from the B register. Thus if 'addr' is specified, it should be one greater than the last item in the choice point to be printed.

e [addr]
: Print the environment at 'addr'. If 'addr' is not specified, the value is taken from the E register.

ex [addr]
: Print the extended environment at 'addr'. If 'addr' is not specified, the value is taken from the E register.

p id Print the state of process #id.

pc a,b
: Print the contents of the code space from 'a' to 'b'.

pd a,b
: Print the contents of the data space from 'a' to 'b'.

pl Print the label table

pp Print the procedure table.

pr Print the current values of all registers.

ps Print the system status. This shows all processors with running or sleeping processes and their corresponding pid's.

q Quit.

s Single step. Find the next processor with a running process and simulate it for one instruction.

t id Trace execution of the indicated process. When the process executes an instruction, the simulator suspends and enters debug mode. Execution may be resumed with 'c' or 's'. Currently up to eight processes may be traced.

u pid Untrace process #pid.

w val[,pid]
: Write the term represented by 'val' on the console. This is essentially a call to the

Prolog 'write' function. This command should be used carefully, as an improper value for 'val' can cause the simulator to crash. *If a process id is specified*, the value will be written as seen by the specified process. This may be different if, for example, the process is an OR process examining a locally bound variable.

wi pid

Print the binding window for the indicated process.

If, instead of a command, a carriage return is used, the command defaults to the last command entered. Thus 's' followed by a series of returns will step through the system.

Upon query success or goal failure, the simulator will respond with the prompt "quit>". A carriage return here will exit the program, while typing 'd' will return the simulator to debug mode, where the previous commands may be used.

7. The .data and .ggraph Files

If the '-i' option is used, the simulator will produce a .data file containing performance measurements. This file contains numerous instrumentation information, including:

(1) All instrumentation information provided by the PLM *simulator*.

(2) Information indicating the number of processors simulated, and whether the program simulated employed AND-parallelism, OR-parallelism, and/or intelligent backtracking.

(3) The number of context swaps.

(4) The number *of microcycles executed* by each processor (denoted by uCYCLES). Note that this is not a direct measure of execution time, because during execution processors may wait for access to the host or a synchronization lock. Microcycles that read or write may also take longer.

(5) The number of reads and writes performed by each processor (denoted by RDS and WTS).

(6) The number of requests made of the host by each processor (denoted by EF, *for* "external functions"). This includes requests for interprocess communication, process creation, termination, etc.

(7) The number of critical sections entered by each processor (denoted by CSEC).

(8) An estimate of the execution time and the assumptions on which the estimate is based. The execution time for processor p is given by

$$T_p = uCYCLES_p - RDS_p - WTS_p + RDS_p{*}T_r + WTS_p{*}Tw + CSEC{*}T_s + EF{*}T_{ef}$$

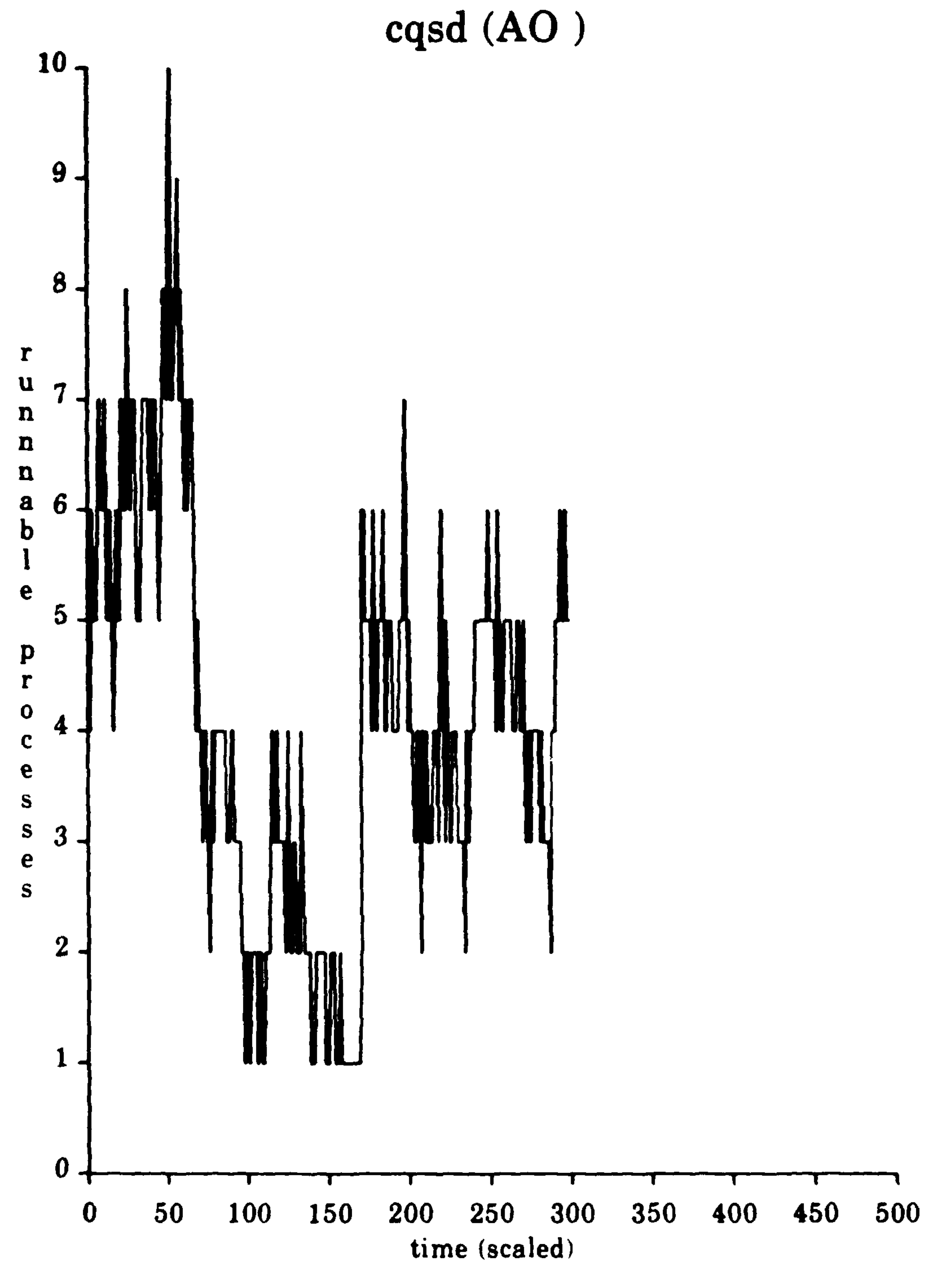
where T_r is the time for a read, T_u the time for a write, T_s the time to get a synchronization lock, and T_{ef} the time it takes for the host to respond to a request from a processor. The execution time for the program is that of the processor with the longest execution time.

(9) The "execution time weights". A processor may spend its time executing internally, reading, writing, accessing the host and waiting for a response, or accessing a synchronization variable. These weights give the ratio of the number of times each of these operations occurred to their sum, to enable the user to estimate the effect of errors in the various parameters on the total execution time.

(10) The number of access *made to the synchronization* memory and the ratio of these to the total number of memory accesses, expressed as a percent.

(11) The average number of runnable and running processes.

If the '-i' option is indicated the simulator will also produce a .ggraph file. This can be used as an input to ggraph to produce an approximate graph of runnable processes versus time. Each time a process executes one instruction, the number of runnable and running processes is recorded in a temporary file. At program termination, this data in this file is compressed into slightly less than 500 data points, since 500 points is the maximum number of points ggraph can process. Typing "ggraph foo.ggraph" will produce the file foo.grn, which can then be printed using gprint or grn and ditroff. The graph will contain the prefix of the benchmark's filename, and a character string containing all or part of the letters "AOI", indicating whether AND-parallelism, OR-parallelism, and/or intelligent backtracking was used by the program. The following is a typical process graph:



8. Compatibility with PLM

The PPP simulator can run all PLM programs. It uses the same data types, and provides all the builtin functions with the exception of 'assert' and 'retract'. Its instruction set includes that of the PLM. The only incompatibility between the two simulators is the syntax of some of the debugging commands: some old PLM commands are no longer supported

9. References

[Dob84] T. P. Dobry, "PLM Simulator Reference Manual", *Working Paper 3.3*, Berkeley, CA, Sep. 84.

[DCD85] T. P. Dobry, J. H. Chang. A. M. Despain and Y. N. Patt, "Extending a Prolog Machine for Parallel Execution", *Proceedings of Hawaii Int. Conf. on System Science 86*, 1985.

[FaD85] B. Fagin and T. Dobry, *The Berkeley PLM Instruction Set: An Instruction Set for Prolog*, Computer Science Division, University of California, Berkeley, Sep 1985. Research Report No UCB Computer Science Dpt. 86/257.

[War83] D. H. D. Warren, *An Abstract Prolog Instruction Set*, Computer Science and Technology Division, SRI, Menlo Park, CA, Oct. 1983. Technical Note 309, Artificial Intelligence Center

10. Appendix: Sample Program, Compilation, and Simulation

The following Prolog program:

```
main :- concat(X,Y,[a,b,c,d,e]),write(X),nl,write(Y),nl,nl,fail.

concat([],L,L).
concat([A|B],C,[A|D]) :- concat(B,C,D).
```

Compiles into:

```
procedure  main/0

_332:
        put_list  X3
        unify_constant  a
        unify_constant  b
        unify_constant  c
        unify_constant  d
        unify_constant  e
        unify_nil
        allocate
        put_variable  Y2,X1
        put_variable  Y1,X2
        call  concat/3,2
        put_unsafe_value  Y2,X1
        escape  write/1
        escape  nl/0
        put_unsafe_value  Y1,X1
        escape  write/1
        escape  nl/0
          escape  nl/0
        fail

 procedure concat/3
           switch_on_term C1,C2,fail

 C1a:      try_me_else_p 2,C2a
 C1:       get_nil A1
           get_value       A2,A3
           proceed

 C2a:      trust_me_else_p          2
 C2:       get_list A1
           unify_variable X4
           unify_cdr       A1
           get_list A3
           unify_value     X4
           unify_cdr       A3
           execute         concat/3
 end
```

Sample execution run on simulator:

```
Script started on Thu Aug 28 09:16:15 1986
1 [ji] -> ppp -d Benchmarks/Par/ccon6.w
pau load  /a/hprg/fagin/PPP1/Benchmarks/Par/concatOP.w
Cspace = 31
pau load  Benchmarks/Par/ccon6.w
Cspace = 31
dbg> s
Running process 0
CLOC 00000000: put_list       X3
dbg>
Running process 0
CLOC 00000001: unify_constant        a
dbg>
Running process 0
CLOC 00000002: unify_constant        b
dbg>
Running process 0
CLOC 00000003: unify_constant        c
dbg>
Running process 0
CLOC 00000004: unify_constant        d
dbg>
Running process 0
CLOC 00000005: unify_constant        e
dbg>
Running process 0
CLOC 00000006: unify_nil
dbg>
Running process 0
CLOC 00000007: allocate
dbg>
Running process 0
CLOC 00000008: put_variable Y2,X1
dbg>
Running process 0
CLOC 00000009: put_variable Y1,X2
dbg>
Running process 0
CLOC 0000000a: call   concat/3,2
dbg>
Running process 0
CLOC 00000013: switch_on_term        C1,C2,fail
dbg>
Running process 0
CLOC 00000014: try_me_else_p        2,C2a
dbg>
Running process 0
CLOC 00000018: trust_me_else_p      2
Created OR process, parent = 0, id = 1
dbg> ps
0       0       RUNNING
```

```
1      1      RUNNING
dbg> s
Running process 1
CLOC 00000019: get_list       A1
dbg>
Running process 0
CLOC 00000015: get_nil        A1
dbg>
Running process 1
CLOC 0000001a: unify_variable       X4
dbg>
Running process 0
CLOC 00000016: get_value    A2,A3
dbg>
Running process 1
CLOC 0000001b: unify_cdr    A1
dbg>
Running process 0
CLOC 00000017: proceed
dbg>
Running process 1
CLOC 0000001c: get_list       A3
dbg>
Running process 0
CLOC 0000000b: put_unsafe_value    Y2,X1
dbg>
Running process 1
CLOC 0000001d: unify_value  X4
dbg>
Running process 0
CLOC 0000000c: escape        write 1
 dbg>
Running process 1
CLOC 0000001e: unify_cdr    A3
dbg>
Running process 0
CLOC 0000000d: escape        nl 0

dbg>
Running process 1
CLOC 0000001f: execute       concat 3
dbg>
Running process 0
CLOC 0000000e: put_unsafe_value    Y1,X1
dbg>
Running process 1
CLOC 00000013: switch_on_term      C1,C2,fail
dbg>
Running process 0
CLOC 0000000f: escape        write 1
a b c d e dbg>
Running process 1
CLOC 00000014: try_me_else_p       2,C2a
```

```
dbg>
Running process 0
CLOC 00000010: escape        nl/0

dbg>
Running process 1
CLOC 00000018: trust_me_else_p     2
Created OR process, parent = 1, id = 2
dbg> ps
0    0    RUNNING
1    1    RUNNING
2    2    RUNNING
dbg> p 0
PROCESS 0
parent = ffffffff; kind = AND; c0 = FORWARD; c1 = ffffffff; state = RUNNING
AX[1] = 00000080;    00000080;    00000080;    9fffffff;
AX[5] = 9fffffff;    9fffffff; 9fffffff; 9fffffff;
P = 00000011; CP = 0000000b;    E = 00004000;    B = 0000401a
TR = 00007c02;    H = 00000086;    HB = 00000086;    S = 00000080
N = 00000002; W = 00000000; PDL = 00008000; E_FF = 00000000
cut = 00000000; ictr = 00000007; parB = 00000000; i = 00000000
Wbase = 00000000; HPbase = 00000080; STKbase = 00004000; TRLbase = 00007c00
dbg> p 1
PROCESS 1
parent = 00000000; kind = OR; c0 = FORWARD; c1 = ffffffff; state = RUNNING
AX[1] = 80008041;    80004004;    00000081;    80008040;
AX[5] = 9fffffff;    9fffffff; 9fffffff; 9fffffff;
P = 00000015; CP = 00000000;    E = 000083ff; B = 00008414
TR = 00008800;    H = 00008042;    HB = 00008042;    S = 00000081
N = 00000000; W = 00008002; PDL = 00008880; E_FF = 00000000
cut = 20000000; ictr = 0000000a; parB = 0000401a; i = 00000002
Wbase = 00008000; HPbase = 00008040; STKbase = 00008400; TRLbase = 00008800
dbg> p 2
PROCESS 2
parent = 00000001; kind = OR; c0 = FORWARD; c1 = ffffffff; state = RUNNING
AX[1] = 80008041;    80004004;    00000081;    80008040;
AX[5] = 9fffffff;    9fffffff; 9fffffff; 9fffffff;
P = 00000019; CP = 00000000;    E = 00008c7f; B = 00008c80
TR = 00009080;    H = 000088c0;    HB = 000088c0;    S = 000088c0
N = 00000000; W = 00008880; PDL = 00009100; E_FF = 00000000
cut = 00000000; ictr = 00000000; parB = 00008414; i = 00000002
Wbase = 00008880; HPbase = 000088c0; STKbase = 00008c80; TRLbase = 00009080
dbg> s
Running process 2
CLOC 00000019: get_list      A1
dbg>
Running process 0
CLOC 00000011: escape        nl/0

dbg>
Running process 1
CLOC 00000015: get_nil       A1
dbg>
```

```
Running process 2
CLOC 0000001a: unify_variable       X4
dbg>
Running process 0
CLOC 00000012: fail
   backtracking...
   restoring from OR choice point
   now listening to p1
   putting p0 to sleep
dbg>
Running process 1
CLOC 00000016: get_value      A2,A3
dbg>
Running process 2
CLOC 0000001b: unify_cdr      A1
dbg>
Running process 1
CLOC 00000017: proceed
   p1 sending SUC to p0
   putting p1 to sleep
dbg>
Running process 2
CLOC 0000001c: get_list       A3
dbg>
Running process 0
CLOC 0000000b: put_unsafe_value     Y2,X1
dbg>
Running process 2
CLOC 0000001d: unify_value  X4
dbg>
Running process 0
CLOC 0000000c: escape         write/1
a dbg>
Running process 2
CLOC 0000001e: unify_cdr      A3
dbg>
Running process 0
CLOC 0000000d: escape         nl/0

dbg>
Running process 2
CLOC 0000001f: execute        concat/3
dbg>
Running process 0
CLOC 0000000e: put_unsafe_value     Y1,X1
dbg>
Running process 2
CLOC 00000013: switch_on_term       C1,C2,fail
dbg>
Running process 0
CLOC 0000000f: escape         write/1
b c d e dbg>
Running process 2
```

```
CLOC 00000014: try_me_else_p        2,C2a
dbg>
Running process 0
CLOC 00000010: escape        nl/0

dbg>
Running process 2
CLOC 00000018: trust_me_else_p    2
Created OR process, parent = 2, id = 3
dbg>
Running process 3
CLOC 00000019: get_list       A1
dbg>
Running process 0
CLOC 00000011: escape        nl/0

dbg>
Running process 2
CLOC 00000015: get_nil        A1
dbg>
Running process 3
CLOC 0000001a: unify_variable        X4
dbg>
Running process 0
CLOC 00000012: fail
  backtracking...
  restoring from OR choice point
  p0 sending NA to p1
. putting p0 to sleep
dbg>
Running process 1 BACKWARD
  backtracking...
  restoring from OR choice point
  now listening to p2
  putting p1 to sleep
dbg> ps
0      0      SLEEPING
1      1      SLEEPING
2      2      RUNNING
3      3      RUNNING
dbg>
0      0      SLEEPING
1      1      SLEEPING
2      2      RUNNING
3      3      RUNNING
dbg>
0      0      SLEEPING
1      1      SLEEPING
2      2      RUNNING
3      3      RUNNING
dbg> s
Running process 2
CLOC 00000016: get_value    A2,A3
```

```
dbg>
Running process 3
CLOC 0000001b: unify_cdr     A1
dbg>
Running process 2
CLOC 00000017: proceed
  p2 sending SUC to p1
  putting p2 to sleep
dbg> c
a b
c d e

a b c
d e

a b c d
e

a b c d e


goal failure
goal failure
Stopped.
cpu time is 3.840000 sec
2 [ji] -> ^D
script done on Thu Aug 28 09:18:16 1986
```

```
/*  file Dspaceutils.c  */
/*  seq   1a.4.3         */

#include "typvars.h"
#include "regs.h"
#include "spacevars.h"
#include "instvars.h"
#include "parameters.h"
#include "timing.h"

#define TMASK 0xf0000000
#define STMASK 0x18000000
#define NUMASK 0x07ffffff
#define CPTMASK 0xc0000000
#define EFFMASK 0x40000000
#define RP(x)  (RDS[PROCESSOR]  += (x))
#define WP(x)  (WTS[PROCESSOR]  += (x))

type( var )
   unsigned long var;

{  return( (TMASK & var) >> 30);  }

cptype( var)
   unsigned long var;
{
return(CPTMASK & var);
}

subtype ( var )
   unsigned long var;

{  return( ( STMASK & var ) >> 27 );  }

long value( var )
   long var;

{  unsigned long i;

   i = ( var << 4 ) >> 4;
   return( i );                }


long numvalue( var )
   long var;

{  long i;

   i = ( var << 6 ) >> 6;
   return( i );                        }


long tag( val, typ )
   unsigned long val;
   int typ;

{
   return((( val << 3 ) >> 3 ) | ( typ << 30 ));    }

long stuck( loc )
  long loc;

{
#ifndef micro
   mixtbl[READS]++;
#endif
   RP(1);
   return( stack[ loc ] );    }

stick( loc, val )
  long loc;
  long val;

{
#ifndef micro
   mixtbl[WRITS]++;
#endif
   WP(1);
   stack[ loc ] = val;  }

effval(e)
long e;
{
if (e & EFFMASK)
   return(1);
return(0);
}

long emask(e_ff)
short e_ff;
{
if (e_ff == 1)
   return(EFFMASK);
return(0);
}
```

```
/* file andp.c */

#include "regs.h"
#include "structures.h"
#include "routines.h"
#include "spacevars.h"
#include "ib.h"
#include "cp.h"
#include "timing.h"

extern short ANDFLAG;
call_p(args)    /* call_p proc/arity, goal_#, jt# */
struct strng args;
{
struct strng proc;
short jt,i;
long a,x,y,ch,p,j,t;

ANDFLAG = 1;
x = delimit( args, ',' );
if( x == 0 ) NOP();
proc = substr( args, 1, x-1 );
args = substr( args, x+1, length(args)-x );
x = delimit( args, ',' );
y = getnum(substr( args, 1, x-1 ));
args = substr( args, x+1, length(args)-x );
jt = getnum( args );
x = locate( Ptbl, Pno, proc );
if( x < 0 ) NOP();
if( y < 0 ) NOP();
if ((scheck == 1) && SAT) {
   CP = P;
   cut = 0;
   P = Ptbl[x].addr;
   j = stuck(E+J_TBL);
   t = stuck(j+jt-1);
   stick(j+jt-1,t | SEQFLAG);
   return;
   }
TP(39);
if (E < B) a = B + scp;
   else a = E + N + envp + scp;

for( i = 0; i < Rn; i++ ) stick(a-i-5, AX[i]);
stick(a-sBCE,E|emask(E_FF));
stick(a-sBCP,P);
stick(a-sB,B | andcp);
stick(a-sM,y);
stick(a-sTR,TR);
stick(a-sH,H);
stick(a-sN,N);
stick(a-sW,W);
HB = H;
B = a;
stick(stuck(E+CP_TBL)+y-1,UNKNOWN);
stick(E+LB,B);
ch = OS_FORK(cpid,AND,Ptbl[x].addr,a,jt);
p = stuck(E+P_TBL);
stick(p+y-1,ch);
j = stuck(E+J_TBL);
t = stuck(j+jt-1);
stick(j+jt-1,t & ~SEQFLAG);
}

d_call_p(args)  /* call_p proc/arity, goal_#, jt# */
struct strng args;
{
struct strng proc;
short jt,i;
long a,x,y,ch,p,j,t;

ANDFLAG = 1;
x = delimit( args, ',' );
if( x == 0 ) NOP();
proc = substr( args, 1, x-1 );
args = substr( args, x+1, length(args)-x );
x = delimit( args, ',' );
y = getnum(substr( args, 1, x-1 ));
args = substr( args, x+1, length(args)-x );
jt = getnum( args );
x = locate( Ptbl, Pno, proc );
if( x < 0 ) NOP();
if( y < 0 ) NOP();
if ((scheck == 1) && SAT) {
   CP = P;
   cut = 0;
   P = Ptbl[x].addr;
   j = stuck(E+J_TBL);
   t = stuck(j+jt-1);
   stick(j+jt-1,t | SEQFLAG);
   return;
   }
TP(20);
if (E < B) a = B + scp;
   else a = E + N + envp + scp;

stick(a-sBCE,E|emask(E_FF));
stick(a-sM,y);
stick(a-sB,B | d_andcp);
B = a;
```

```
stick(stuck(E+CP_TBL)+y-1,NONE);
stick(E+LB,B);
ch = OS_FORK(cpid,DET_AND,Ptbl[x].addr,a,jt);
p = stuck(E+P_TBL);
stick(p+y-1,ch);
j = stuck(E+J_TBL);
t = stuck(j+jt-1);
stick(j+jt-1,t & ~SEQFLAG);
}

wait(arg)
struct strng arg;
{
long l;
GET_LOCK;
l = jte(E, getnum(arg));
if ((l != 0) && !SEQ(l)) {
   RELEASE_LOCK;
   OS_SLEEP(cpid);
   }
else
   RELEASE_LOCK;
}
```

```
/* file basics.c */

#include "regs.h"
#include "instvars.h"
#include "routines.h"
#include "osdefs.h"
#include "typvars.h"
#include "spacevars.h"
#include "parameters.h"
#include "mem.h"
#include "timing.h"
#include "ib.h"
#include "cp.h"
#include "orp.h"
#include "msg.h"

trail( var )
   long var;

{
   TP(4);
   if ((extr(var)) || (( var < HB ) || (( var > H ) && ( var < B ))))
   {  stick(TR++, var);
      mixtbl[TRALS]++;              };   }


short kind;
long windowed();
long dereference( var )
   long var;

{
long val;
switch (kind) {
   case AND:
   case DET_AND:
      TP(2);
      if( type(var) == tvar ) {
         mixtbl[DERFS]++;
         TP(4);
         var = stuck(value(var));
         for( ;
         ((type(var)==tvar)&&(value(var)!=value(stack[value(var)])));
         var = stuck(value(var)) ) {
            mixtbl[DERFS]++;
            TP(4);
            }
         }
      return( var );
      break;
   case OR:
      TP(1);
      if (type(var) == tvar) {
         while (1) {
            if (extr(var))
               break;
            if ((var == stuck(value(var))) || (type(var) != tvar)) {
               TP(5);
               break;
               }
            TP(6);
            mixtbl[DERFS]++;
            var = stuck(value(var));
            }
         if ((type(var) != tvar) || (var == stuck(value(var))))
            return(var);
         if ((val = windowed(var)) > 0)
            return(val);
         }
      return(var);
      break;
   default:
         printf("dereference: error\n");
   }
}


long windowed(var)
long var;
{
long i;
TP(2);
for (i = Wbase; i < W; i+=2) {
   TP(7);
   if (stuck(i) == var) {
      TP(3);
      return(stuck(i+1));
      }
   TP(1);
   }
TP(2);
return(0);
}

long decdr( val )
   long val;

{  S++;
   TP(3);
   if(( val & tcdr ) && ( type(val) == tlst ))
   {  S = value(val);
      val = stuck(S++);  }
   else if(( val & tcdr ) && ( type(val) != tvar ))
      val = tag( tcon, NIL ) | tcdr;
   return( val );   }

extr(var)
long var;
{
TP(3);
if (value(var) >= HPbase) {
   TP(4);
   if (value(var) <= HPbase + PMSIZE)
      return(0);
   }
return(1);
}

kp(b)
long b;
{
long e,p,m,n,id;
short i;
switch (cptype(stuck(b-sB))) {
   case(andcp):
   case(d_andcp):
      GET_LOCK;
      e = numvalue(stuck(b-sBCE));
      p = stuck(e+P_TBL);
      m = stuck(b-sM);
      OS_SEND(cpid,KILL,ID(stuck(p+m-1)));
      RELEASE_LOCK;
      break;
   case(orcp):
      GET_LOCK;
      n = stuck(b-soL);
      TP(4);
      for (i = 1; i < n; i++) {
         id = stuck(b-ocp-i);
         if (!(SEQ(id))) {
            TP(4);
            if (id != -1)
               OS_SEND(cpid,KILL,ID(id));
            }
         }
      RELEASE_LOCK;
   }
}
```

```
/*  file  bind.c  */
/*  seq   1a.3.10  */

#include "regs.h"
#include "structures.h"
#include "typvars.h"
#include "routines.h"
#include "instvars.h"
#include "parameters.h"
#include "process.h"
#include "mem.h"
#include "timing.h"
#include "cp.h"

extern int mixtbl[];
extern struct process *pt[];

bind( binding, bound )
   long binding,bound;

{
long agin,temp;

mixtbl[BINDS]++;
if ((kind == OR) && extr(bound))
   window(bound, binding);
else {
   if ( bound & tcdr )
      stick( value(bound), binding | tcdr);
   else
      stick(value(bound), binding);
   TP(1);
   trail( value(bound) );
   }
}

window(bound, binding)
long bound, binding;
{
TP(5);
stick(W++,bound);
stick(W++,binding);
if (W > HPbase) {
   printf("Window overflow, process %x\n", cpid);
   debug("quit");
   return;
   }
if (W - Wbase > mixtbl[MXWIN])
   mixtbl[MXWIN] = W - Wbase;
}

pb(ch)
long ch;
{
long i,l,binding,bound,OS_GETWBASE(),OS_GETW();
i = OS_GETWBASE(ID(ch));
l = OS_GETW(ID(ch));
TP(4);
while (i < l) {
   TP(9);
   bound = dereference(stuck(i++));
   binding = stuck(i++);
   bind(binding,bound);
   }
}
```

```
/*  file  controll.c  */
/*  seq   1.3.0       */

#include "spacevars.h"
#include "regs.h"
#include "routines.h"
#include "ib.h"
#include "env.h"
#include "timing.h"

allocate()

{  long ce;

   ce = E;
   if( ce < B ) {
      E = B;
      TP(7);
      }
   else {
      E = ce + N + envsz();
      TP(9);
      }
   stick(E + sCP, CP);
   stick(E+sCE, ce|emask(E_FF));
   stick(E + sCN,  N);
   stick(E + sCB, B | cut );
   E_FF = 0;
}

deallocate()
{
   long x;

   TP(5);
   CP = stuck(E+sCP);
   N = stuck(E+sCN);
   x = stuck(E+sCE);
   if (E_FF == 1) {
      TP(2);
      stick(E+sCE,x|EXIT);
      TP(1);
      }
   E_FF = effval(x);
   E = numvalue(x);
}
```

```
/*  file  control2.c  */
/*  seq   1.3.1       */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "routines.h"
#include "msg.h"
#include "timing.h"

extern int quit;
call(args)
   struct strng args;

{  int x,y;
   struct strng proc;

   TP(1);
   x = delimit( args, ',' );
   if( x == 0 ) {
      printf("call: invalid format %s\n", args);
      exit(1);
      }
   else  {
      proc = substr( args, 1, x-1 );
      args = substr( args, x+1, length(args)-x );
      y = getnum( args );
      x = locate( Ptbl, Pno, proc );
      if( y < 0 ) NOP();
      else {
         N = y;
         CP = P;
         cut = 0;
         if( x < 0 )
            fail();
         else
            P = Ptbl[x].addr;
         }
      }
}

execute(args)
   struct strng args;

{  int x;

   TP(1);
   x = locate( Ptbl, Pno, args );
   if( x >= 0 )
```

```
      P = Ptbl[x].addr;
   else if(( x = locate( Ltbl, Lbloc, args )) >= 0 )
      P = Ltbl[x].addr;
   else
      fail();
   cut = 0;
}


proceed()

{
TP(1);
cut = 0;
P = CP;
if ( P == 0 ) {
   if (cpid == 0) {
      printf("Top level query success\n");
      pquit();
      }
   else {
      OS_SEND(cpid,SUC,parent);
      OS_SLEEP(cpid);
      }
   }
}
```

```
/*  file  debug.c   */
/*  seq   1.2.4     */

#include <stdio.h>
#include <signal.h>
#include "regs.h"
#include "structures.h"
#include "spacevars.h"
#include "routines.h"
#include "osdefs.h"
#include "process.h"
#include "parameters.h"

int UPT = -1;
int BKPT = -1, BKPROC = -1, Trace = 0;
extern int quit,silent,dbg,bkflag;
extern FILE *fp;
extern long *processors;

struct strng PR = { 2, " pr" };
struct strng SS = { 1, " s" };
struct strng CO = { 1, " c" };
extern struct strng BLANK;
extern struct process *pt[];

debug( prompt )
   char *prompt;

{  struct strng cmd, line;
   int x,pause;
   long fm,to;
   struct strng inputstr();
   struct process *p;
   char getch();
   static struct strng last = { 1, " c" };

   pause = silent;
   while( pause )
   {  printf("%s> ", prompt);
      line = inputstr( fp );
      if( length( line ) == 0 )
         line = last;
      else
         last = line;
      fm = -1;  to = -1;
      x = delimit( line, ' ' );
      if( x == 0 ) {  cmd = line;  line = BLANK; }
      else  {
         cmd = substr( line, 1, x-1 );
```

```
            line = substr( line, x+1, length(line)-x );
            };
        if (length(line) > 0) {
            x = delimit( line, ',' );
            if( x != 1 ) fm = gethex( line );
            if( x != 0 ) {
                line = substr( line, x+1, length(line)-x );
                to = gethex( line );
                };
            };
        if ((*prompt == 'q') || (*prompt == 'i')) {
            switch( getch( cmd, 1 ) ) {
                case 'd' : prompt = "dbg";
                           quit = 0;
                           break;
                case 'q' : quit = 1;
                           pause = 0;
                           break;
                default  : pause = 0;
                }
            }
        else {
        switch( getch( cmd, 1 ) ) {
            case 'w' : switch(getch(cmd,2)) {
                           case 'i':
                               Wprn(fm);
                               break;
                           default:
                               if (to != -1) {
                                   p = pt[to];
                                   kind = p -> kind;
                                   Wbase = p -> Wbase;
                                   W = p -> W;
                                   HPbase = p -> HPbase;
                                   }
                               dwrite(fm);
                               printf("\n");
                               break;
                           }
                       break;
            case 'e' : switch(getch(cmd,2)) {
                           case 'x': EXprn(fm);
                                     break;
                           default : Eprn(fm);
                                     break;
                           }
                       break;
            case 'p' : poke(cmd,fm,to);
                       break;
            case 'b' : BKPT = fm;
```

```
                        BKPROC = to;
                        break;
            case 's' :
                        dbg = 1;
                        cont(0);
                        break;
            case 'c' :
                        switch(getch(cmd,2)) {
                           case 'p' :
                              CPprn(fm);
                              break;
                           default :
                              dbg = 0;
                              cont(1);
                              break;
                           }
                        break;
            case 't' :  tproc(fm);
                        break;
            case 'u' :  utproc(fm);
                        break;
            case 'q' :  quit = 1;
                        pause = 0;
                        break;
            default   : printf("Unknown command ");
                        outputstr( cmd );
                        printf("\n");
            }
            if (quit)
               pause = 0;
            }
      }
}

#define SIM 0
#define MSGS 1

cont(x)
int x;
{
static lp = -1;
while (1) {
   for (PROCESSOR = lp+1; PROCESSOR < PROCESSORS; PROCESSOR++) {
      if ((processors[PROCESSOR] != -1) &&
      (pt[processors[PROCESSOR]] -> state == RUNNING)) {
         sim();
         lp = PROCESSOR;
         if ((x == 0) || quit)
            return;
         if (bkflag) {
                bkflag = 0;
                return;
                }
            }
        }
    if (PROCESSOR == PROCESSORS)
        lp = -1;
    }
}

intrpt( no )
    int no;

{  printf( "\nInterrupt...\n" );
   signal( SIGINT, intrpt );
   bkflag = 1;
}
```

```
/*  file  debug2.c  */
/*  seq   1.2.5     */

#include <stdio.h>
#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "parameters.h"
#include "typvars.h"
#include "routines.h"
#include "osdefs.h"
#include "msg.h"

#ifdef micro
extern int UPT;
#endif
extern int BKPT, Trace, quit,inst_total;
extern int lP;
extern FILE *fp;
FILE *fopen();
static struct strng psestr = { 5, " pause" };
extern struct strng f_name,name;

go( fm, to )
   long fm,to;

{
#ifdef micro
   if( uPC == -2 )
#endif
   lP = P;
   if( fm >= 0 ) { printf("CAUTION: jumping out of normal sequence. \n");
                   P = fm;  };
   while(( !quit ) && (( to==-1 )||( P!=to )) &&
#ifdef micro
                   (( uPC != UPT ) || ( to < -1 )) &&
#endif
                   (( lP!=BKPT ) || ( to < -1 )) && ( to!= -4 ))
   {  if( equalstr( Cspace[P].inst, psestr )) to++;
#ifdef micro
      if( uPC == -2 )
#endif
      lP = P;
      dispatch();
      if( Trace ) Cprn( lP,lP );
      if( to < -1 ) to--;      };
   if(( !Trace ) && ( !quit )) Cprn( lP, lP );
   else Trace = 0;                 }
```

```
poke( cmd, fm, to )
   struct strng cmd;
   long fm, to;

{
   if ( to < 0 )
      to = fm;
   switch( getch( cmd, 2 )) {
      case 'c' : Cprn( fm, to );
                 break;
      case 'd' : Dprn( fm, to );
                 break;
      case 'p' : Pprn( 1, Pno );
                 break;
      case 'l' : Lprn( 1, Lbloc );
                 break;
      case 'r' : Rprn();
                 break;
      case 's' : Sprn();
                 break;
      default  : PTprn(fm);
                 break;
      }
}

reset( line )
   struct strng line;

{  int i;

   i = delimit( line, ',' );
   if( i > 0 )
   {  if(( getch( line, i-1) == 'i' ) && ( inst_total != 0 ))
         prntmix( f_name );
      line = substr( line, i+1, length(line)-i );  };
   initmix();
   name.len = 0;
   f_name = line;
   lP = 0;
   quit = 0;
   BKPT = -1;
   P = 0;  CP = 0;  E = STKbase;  B = STKbase;  W = 0;
   TR = TRLbase;  H = HPbase;  HB = HPbase;  S = HPbase;
#ifndef micro
   PDL = Dsiz;
#else
   PDL = 0;
#endif
   N = 0;  mode = 0;
   for( i = 0; i < Rsiz; i++ ) AX[i] = tag ( -1, tvar );
```

```
#ifdef micro
   uPC = -2;
   prefetch(1);
#endif
   if( length(line) != 0 )
   {  Caddr = 0;
      Pno = 0;
      Lbloc = 0;
      Cno = -1;
      load( line );   };
   fp = stdin;                                         }

int tr[TPROCS];
tracing(p)
long p;
{
short i;
for (i = 0; i < TPROCS; i++)
   if (tr[i] == p)
      return(1);
return(0);
}

tproc(p)
long p;
{
short i;
for (i = 0; i < TPROCS; i++) {
   if (tr[i] == -1) {
      tr[i] = p;
      return;
      }
   }
if (i == TPROCS)
   printf("Can't trace any more processes\n");
}

utproc(p)
long p;
{
short i;
for (i = 0; i < TPROCS; i++)
   if (tr[i] == p)
      tr[i] = -1;
}


dwrite(l)
long l;
{  long val,ptr,temp;
```

```
    struct strng const;

    val = dereference(1);
    if(( type( val ) == tlst ) || ( type( val ) == tstr ))
    {  if( val & tcdr ) ptr = NIL;
       else  {
       temp = stuck(value(val)+1);
       if( temp & tcdr)
          if( value(temp) == NIL )
             ptr = NIL;
          else ptr = value(temp);
       else ptr = value(val)+1;
       val = stuck(value(val));    };  }
    else  ptr = NIL;
    while( ptr != NIL )
    {  printer( val );
       val = stuck(value(ptr));
       if( val & tcdr ) ptr = NIL;
       else {
       temp = stuck(ptr+1);
       if( temp & tcdr)
          if( type(temp) == tcon )
             ptr = NIL;
          else ptr = value(temp);
       else ptr = ptr+1;  };  };
    printer( val );
    if(( type(temp) == tcon ) && ( temp != NIL )) printer( temp );   }
```

```
/*  file  dispatch.c  */
/*  seq   1.2.3       */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "parameters.h"
#include "instvars.h"
#include "osdefs.h"
#include "cp.h"
#include "orp.h"
#include "timing.h"

long c0,c1;
extern int quit;
extern int fl,esc,lP, mrk;
extern long ictr;
extern struct tbl decode_tbl[];

dispatch()
{  long x;
   long i;

   inst_total++;
   ictr++;
   if(( H>STKbase ) || ( E>TRLbase ) || ( B>TRLbase ) || ( TR>PDL )){
      if (H > STKbase)
         printf("HEAP ");
      else if ((E > TRLbase) || (B > TRLbase))
         printf("STACK ");
      else
         printf("TRAIL ");
      printf("OVERFLOW, PROCESS %x\n", cpid);
      printf("HPbase = %08x, STKbase = %08x, TRLbase = %08x\n",
      HPbase,STKbase,TRLbase);
      Rprn();
      debug("quit");
      quit = 1;
      }
   if( H - HPbase > mixtbl[MXHEP] ) mixtbl[MXHEP] = H - HPbase;
   if( E - STKbase > mixtbl[MXSTK] ) mixtbl[MXSTK] = E - STKbase ;
   if( B - STKbase > mixtbl[MXSTK] ) mixtbl[MXSTK] = B - STKbase ;
   if( TR - TRLbase > mixtbl[MXTRL] ) mixtbl[MXTRL] = TR - TRLbase;
   fl = 0; esc = 0; lP = P; mrk = 0;
   if (c0 != FORWARD)
      i = c0;
   else {
      x = P++;
      i = locate( decode_tbl, NUMINSTS, Cspace[x].inst );
```

```
       }
    switch( i )
    {
    case -1:  NOP();                                      break;
    case  0:  proceed();                                  break;
    case  1:  execute( Cspace[x].args );                  break;
    case  2:  call( Cspace[x].args );                     break;
    case  3:  allocate();                                 break;
    case  4:  deallocate();                               break;
    case  5:  get_variable(Cspace[x].args);               break;
    case  6:  get_value(Cspace[x].args);                  break;
    case  7:  get_constant(Cspace[x].args);               break;
    case  8:  get_nil(Cspace[x].args);                    break;
    case  9:  get_structure(Cspace[x].args);              break;
    case 10:  get_list(Cspace[x].args);                   break;
    case 11:  put_variable(Cspace[x].args);               break;
    case 12:  put_value(Cspace[x].args);                  break;
    case 13:  put_unsafe_value(Cspace[x].args);           break;
    case 14:  put_constant(Cspace[x].args);               break;
    case 15:  put_nil(Cspace[x].args);                    break;
    case 16:  put_structure(Cspace[x].args);              break;
    case 17:  put_list(Cspace[x].args);                   break;
    case 18:  unify_void(Cspace[x].args);                 break;
    case 19:  unify_variable(Cspace[x].args);             break;
    case 20:  unify_local_value(Cspace[x].args);          break;
    case 21:  unify_value(Cspace[x].args);                break;
    case 22:  unify_constant(Cspace[x].args);             break;
    case 23:  unify_nil();                                break;
    case 24:  try_me_else(Cspace[x].args);                break;
    case 25:  retry_me_else(Cspace[x].args);              break;
    case 26:  trust_me_else(Cspace[x].args);              break;
    case 27:  try(Cspace[x].args);                        break;
    case 28:  retry(Cspace[x].args);                      break;
    case 29:  trust(Cspace[x].args);                      break;
    case 30:  switch_on_term(Cspace[x].args);             break;
    case 31:  switch_on_constant(Cspace[x].args);         break;
    case 32:  switch_on_structure(Cspace[x].args);        break;
    case 33:  pquit();                                    break;
    case 34:  fail();                                     break;
    case 35:  Rprn(); outputstr(Cspace[x].args);
              printf(" "); debug( "pause" );              break;
    case 36:  unify_cdr(Cspace[x].args);                  break;
    case 37:  mark(Cspace[x].args);                       break;
    case 38:  sys_escape(Cspace[x].args);                 break;
    case 39:  cutp();                                     break;
    case 40:  cutd(Cspace[x].args);                       break;
    case 41:  make(Cspace[x].args);                       break;
    case 42:  enter(Cspace[x].args);                      break;
    case 43:  i_allocate(Cspace[x].args);                 break;
    case 44:  i_cut(Cspace[x].args);                      break;
```

```
   case 45:  i_call(Cspace[x].args);                  break;
   case 46:  call_p(Cspace[x].args);                  break;
   case 47:  wait(Cspace[x].args);                    break;
   case 48:  try_p(Cspace[x].args);                   break;
   case 49:  retry_p(Cspace[x].args);                 break;
   case 50:  trust_p(Cspace[x].args);                 break;
   case 51:  cut_p();                                 break;
   case 52:  try_me_else_p(Cspace[x].args);           break;
   case 53:  retry_me_else_p(Cspace[x].args);         break;
   case 54:  trust_me_else_p(Cspace[x].args);         break;
   case 55:  kill_desc_and_die();                     break;
   case 56:  fail_from_or();                          break;
   case 57:  fail_from_and();                         break;
   case 58:  backward();                              break;
   case 59:  d_call_p(Cspace[x].args);                break;
                                                                };
   if ((ictr % MAXINSTS) == 0) {
      OS_TIMEOUT(cpid);
      ictr = 0;
      }
   c0 = FORWARD;
   mixtbl[i]++;
}

backward()
{
fail();
}

fail_from_and()
{
long parB = c1;
while (B != parB) {
   TP(5);
   kp(B);
   B = numvalue(stuck(B - sB));
   }
B = numvalue(stuck(B - sB));
fail();
}

fail_from_or()
{
fail();
}

kill_desc_and_die()
{
while (B != STKbase) {
   TP(5);
```

```
        kp(B);
        B = numvalue(stuck(B - sB));
        }
    GET_LOCK;
    EF[PROCESSOR]++;
    OS_DIE(cpid);
    RELEASE_LOCK;
    }
```

```
/*  file  escape.c  */
/*  seq   1.3.17    */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "instvars.h"

int esc;
extern struct tbl bi_tbl[];
extern int fl;

sys_escape( args )
   struct strng args;

{  int i,sp;
   struct strng proc;

   mixtbl[ESCPS]++;
   esc = 1;  sp = 0;
   i = delimit( args, ',' );
   if( i == 0 ) i = length(args)+1;
   proc = substr( args, 1, i-1 );
   i = locate( bi_tbl, 40, proc );
   switch( i )
   {  case  0 :
      case  1 :
      case  2 :
      case  3 :
      case  4 :
      case  5 :
      case  6 :
      case  7 :
      case  8 :
      case  9 :
      case 10 : compare(i, args );                  break;
      case 11 :
      case 12 : is( args );                         break;
      case 13 :
      case 14 : writer( args );                     break;
      case 15 :
      case 16 : nl();                               break;
      case 17 : sp = 1; caller();                   break;
      case 18 :
      case 19 : var();                              break;
      case 20 :
      case 21 : setter();                           break;
```

```
        case 22 :
        case 23 : access();                                  break;
        case 24 :
        case 25 : integer();                                 break;
        case 26 : asserta();                                 break;
        case 27 : retracta();                                break;
        case 28 : retractp();                                break;
        case 29 : assertz();                                 break;
        case 30 : stris();                                   break;
        case 31 : rplac();                                   break;
        case 32 : nrplac();                                  break;
        case 33 : rplacd();                                  break;
        case 34 : eq();                                      break;
        default : printf("%s: unknown builtin\n".proc.ch);
                  fail();                                  };
                  }
```

```
/*  file  escape0.c  */
/*  seq   1.3.21     */

#include "regs.h"
#include "structures.h"
#include "typvars.h"
#include "routines.h"
#include "instvars.h"
#include "parameters.h"

extern int mixtbl[];

esc_unify( arg1, arg2 )
   long arg1,arg2;

/*  {  return( unify( arg1, arg2 ));  }  */
{  long temp;

   mixtbl[UNIFS]++;  mixtbl[UNIFR]++;
   if(( type( arg1 ) == tvar ) || ( type ( arg2 ) == tvar ))
   {  if(( type ( arg1 ) == tvar ) && ( type(arg2) == tvar))
         if(value(arg1) < value(arg2))
            esc_bind( arg1, arg2 );
         else esc_bind( arg2, arg1 );
       else if( type(arg1) == tvar)  esc_bind( arg2, arg1 );
            else  esc_bind( arg1, arg2 );
       return( 1 );                                          }
   else if(( type(arg1) == tcon ) && ( arg1 == arg2)) return( 1 );
   else if((( type(arg1) == tlst ) && ( type(arg2) == tlst )) ||
           (( type(arg1) == tstr ) && ( type(arg2) == tstr )))
   {  mixtbl[UNIFS]--;
      if( !esc_unify( stuck(value(arg1)), stuck(value(arg2)) )) return(0);
      else
      {  temp = value(arg1)+1;
         arg1 = stuck( temp );
         if(!( arg1 & tcdr )) arg1 = tag( temp, tlst );
         temp = value(arg2)+1;
         arg2 = stuck( temp );
         if(!( arg2 & tcdr )) arg2 = tag( temp, tlst );
         if( !esc_unify( arg1, arg2 ))
             return(0);
         else return(1);     };                                }
   else return( 0 );                                              }


esc_bind( binding, bound )
   long binding,bound;

{  long agin,temp;
if(( type(binding) == tvar ) || ( type(binding) == tcon ))
{  if( bound & tcdr )
      stick( value(bound), binding | tcdr);
   else
      stick(value(bound), binding);
/*   trail( value(bound) );  */
   }
else
{  if( type(bound) == tcdr )
      stick( value(bound), tag(H,tcdr));
   else
      stick(value(bound), tag( H, type(binding)));
 /*  trail( value(bound) );  */
   for( agin = 1; agin == 1; )
   {
      for( S = value(binding); S != NIL; )
      {
      if( type( stuck(S) ) == tvar )
         {  temp = tag( H, tvar );
            stick( H, temp );
            stick( stuck(S), temp );
            H++;  }
         else if( type( stack[S] ) == tcon )
           stick(H++, stack[S]);
         else
         {  stick(--PDL, S );
            stick(--PDL, H);
            stick(H++, tag( -1, type(stack[S]) ));  };
         temp = stuck(++S);
         if( temp & tcdr )
            if(( value(temp) == NIL ) || ( value(temp) == S ))
            {  S = NIL;
               if( value(temp) == NIL ) stick(H++,temp);
               else { stick( H, tag(H,tcdr) ); H++; };   }
            else S = value(temp);
            };
      if( PDL < mixtbl[MXPDL] ) mixtbl[MXPDL] = PDL;
      if( PDL < Dsiz )
      {  temp = stuck(PDL++);
         binding = stuck(PDL++);
         binding = stuck(binding);
         stick(temp, tag( H, type( stack[temp] )));  }
      else agin = 0;  };
                                                   }  }
```

```
/*  file  escape1.c  */
/*  seq   1.3.18     */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "instvars.h"

#define NUMASK 0x07ffffff

compare( i, args )
   int i;
   struct strng args;

{  long val1,val2,stris();
   long n1,n2,temp0;

   val1 = dereference( AX[0] );
   val2 = dereference( AX[1] );
   if( i == 0 )
      if( !unify( val1, val2 ))
         fail();
      else;
   else {
      if (type(val1) == tstr) {
            temp0 = AX[0];
            AX[0] = tag( H, tvar );
            stick( H, AX[0] );
            val1 = stris();
            AX[0] = temp0;
            }
      if (type(val2) == tstr) {
            temp0 = AX[0];
            AX[0] = tag( H, tvar );
            stick( H, AX[0] );
            val2 = stris();
            AX[0] = temp0;
            }
      if ((type(val1) != tcon) || (subtype(val1) != tnum) ||
      (type(val2) != tcon) || (subtype(val2) != tnum))
         fail();
      else {
         n1 = numvalue( val1 );
         n2 = numvalue( val2 );
         switch( i ) {
            case 1 :
                     if (n1 != n2) fail();   break;
            case 2 :
                     if (n1 == n2) fail();   break;
            case 3 :
            case 4 :
                     if (n1 >= n2) fail();   break;
            case 5 :
            case 6 :
                     if (n1 <= n2) fail();   break;
            case 7 :
            case 8 :
                     if (n1 > n2) fail();    break;
            case 9 :
            case 10:
                     if (n1 < n2) fail();    break;
            }
         }
      }
   }

is( args )
   struct strng args;

{  struct strng op,opnd;
   long val1,val2,val0;
   long ans,n1,n2;

   val0 = dereference( AX[0] );
   val1 = dereference( AX[1] );
   val2 = dereference( AX[3] );
   if((( type( val0 ) == tvar ) || ( type( val0 ) == tcon ))
           && ( type( val1 ) == tcon ) &&
      ( type( AX[2] ) == tcon ) && ( type( val2 ) == tcon )) &&
      (( subtype( val2 ) == tnum ) && ( subtype( val1 ) == tnum )))
   {  n1 = numvalue(val1);
      n2 = numvalue(val2);
      switch( getch( Cspace[value(AX[2])].inst, 1 ))
      {  case '+' : ans = n1 + n2;
                    break;
         case '-' : ans = n1 - n2;
                    break;
         case '*' : ans = n1 * n2;
                    break;
         case '/' : ans = n1 / n2;
                    break;
         case '%' : ans = n1 % n2;
                    break;
         case 'm' : if((getch(Cspace[value(AX[2])].inst, 2)=='o') &&
                    (getch(Cspace[value(AX[2])].inst, 3) == 'd') &&
                    (getch(Cspace[value(AX[2])].inst, 4) == '/')) {
                       ans = n1 % n2;
```

```
                        break;
                        }
          default  : printf(" Illegal operation \n" );
                     ans = 0;
                                                    };
       if(( type( val0 ) == tcon ) && ( subtype( val0 ) == tnum ) &&
          ( numvalue( val0 ) != ans )) fail();
       else if( type( val0 ) == tvar )
       {  sprintf( args.ch, " &%010d", ans );
          args.len = 11;
          val1 = get_consts( args );
          bind( val1, AX[0] );                             };        }
    else fail();
                                                    }

long stris()

{  long temp0,temp1,var,val,op,n1,n2;
   char ch;

   temp0 = AX[0];
   temp1 = AX[1];
   var = dereference( AX[0] );
   val = dereference( AX[1] );
   if((( type(var) != tvar ) && ( type(var) != tcon )) ||
      (( type(val) != tcon ) && ( type(val) != tstr )))
      fail();
   else if( type(val) == tstr ) {
      S = value(val);
      op = stuck(S++);
      ch = getch( Cspace[value(op)].inst, 1 );
      n1 = stuck(S++);
      n2 = stuck(S);
      if( type(n1) == tstr ) {
         AX[0] = tag( H, tvar );
         stick( H, AX[0] );
         AX[1] = n1;
         stris();
         n1 = dereference( AX[0] );
         }
      if( type(n2) == tstr ) {
         AX[0] = tag( H, tvar );
         stick( H, AX[0] );
         AX[1] = n2;
         stris();
         n2 = dereference( AX[0] );
         }
      if(( type(n1) != tcon )||( type(n2) != tcon ))
         fail();
      else {
        switch( ch ) {
           case '+' : val = numvalue(n1)+numvalue(n2); break;
           case '-' : val = numvalue(n1)-numvalue(n2); break;
           case '*' : val = numvalue(n1)*numvalue(n2); break;
           case '/' : val = numvalue(n1)/numvalue(n2); break;
           case '%' : val = numvalue(n1)%numvalue(n2); break;
           case 'm' : if((getch(Cspace[value(op)].inst, 2)=='o') &&
                      (getch(Cspace[value(op)].inst, 3) == 'd') &&
                      (getch(Cspace[value(op)].inst, 4) == '/')) {
                         val = numvalue(n1) % numvalue(n2);
                         break;
                      }
           default  : printf("illegal operation \n" );
           val =  0;
           }
        }
     n1 = ( val & NUMASK )  | ( tnum << 27 );
     val = tag( n1, tcon );
     }
  if (!unify(val,var))
     fail();
   else {
     AX[0] = temp0;
     AX[1] = temp1;
     return(val);
     }
}
```

```
/*  file  escape2.c  */
/*  seq   1a.3.19     */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "instvars.h"

nl()

{  printf("\n");  }

writer( arg )
  int arg;

{  long val,ptr,temp;
   struct strng const;
   int i;

   val = dereference( AX[0] );
   i = 0;
   if( type( val ) == tstr ) i = strwrite( val );
   if( !i )  {
   if(( type( val ) == tlst ) || ( type( val ) == tstr))
   {  if( val & tcdr ) ptr = NIL;
      else  {
      temp = stuck(value(val)+1);
      if( temp & tcdr)
         if( temp == NIL )
            ptr = NIL;
         else ptr = value(temp);
      else ptr = value(val)+1;
      val = stuck(value(val));   };  }
   else  ptr = NIL;
   while( ptr != NIL )
   {  printer( val );
      val = stuck(value(ptr));
      if( val & tcdr ) ptr = NIL;
      else {
      temp = stuck(ptr+1);
      if( temp & tcdr)
         if( type(temp) == tcon )
            ptr = NIL;
         else ptr = value(temp);
      else ptr = ptr+1;  };  };
   printer( val );
   if(( type(temp) == tcon ) && ( temp != NIL )) printer( temp );   };
   }


printer( val )
   long val;

{  long item;

   val = dereference( val );
   item = value( val );
   if(( val & tcdr ) && ( type(val) != tlst )) printf(" | ");
   switch( type( val ) )
   {  case tvar : printf(" VARIABLE %08x ", item );       break;
      case tstr : printer( stuck( item ) );
                  printf( "( " );
                  AX[0] = tag( item+1, tlst );
                  writer(1);
                  printf( ") " );
                  break;
      case tlst : printf( "( " );
                  AX[0] = val;
                  writer(2);
                  printf( ") " );
                  break;
      case tcon :
                 if( subtype( val ) == tnum )
                  {  item = numvalue( val );
                     printf(" %d ", item );  }
                  else
                  {  outputstr( Cspace[item].inst );
                     printf(" ");                     };
                                                                 };  }

strwrite( val )
  long val;

{  long stradd,func,temp;
   int n,i;
   struct strng fname,arity;

   stradd = value( val );
   func = stack[stradd];
   if( type( func ) != tcon )  return(0);
   else  {
      fname = Cspace[value(func)].inst;
      arity = fname;
      i = delimit(arity, '/' );
      while( i != 0 ) {
         arity = substr(arity,i+1,length(arity)-i);
         i = delimit(arity, '/' );   };
```

```
        if((equalstr(arity,fname))||(length(arity) == 0)) return( 0 );
        else  {
        outputstr( fname );
        printf( "( " );
        n = getnum( arity );
        for(i = 1; i <= n; i++ )
        {  temp = AX[0];
           AX[0] = stack[stradd+i];
           writer(3);
           AX[0] = temp;    };
        printf(" )");    };    };
    return(1);    }
```

```
/*  file  escape3.c  */
/*  seq   1.3.20     */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"

extern long H2;
long tempH;

caller()

{  int i;
   long val;
   struct strng arg;

   val = dereference( AX[0] );
   for( i = 1; i < Rn; i++ )  AX[i-1] = AX[i];
   if( type( val ) != tcon ) fail();
   else  {
   arg = Cspace[value(val)].inst;
   i = locate( Ptbl, Pno, arg );
   if( i < 0 ) sys_escape( arg );
   else {  CP = P;
           P = Ptbl[i].addr;   };   };
                                 }

var()

{  long val;

   val = dereference( AX[0] );
   if( type(val) != tvar ) fail();   }


setter()

{  long val,var;

   val = dereference( AX[0] );
   var = dereference( AX[1] );
   if(( type( var ) == tcon ) && ( subtype( var ) == tnum ))
   {  if( numvalue(var) >15 ) { printf("Illegal Set variable %d \n",
                                       numvalue(var));
                                printf("Set variable 0 used.\n");
                                var = 0;  };
      stick( numvalue(var), tag( numvalue(var), tvar ));
```

```
        tempH = H;
        H = H2;
        /* What if this unification fails? */
        esc_unify( stack[numvalue(var)], val );
        H2 = H;
        H = tempH;    }
    else fail();
                                                            }

access()

{  long var,var1,val;

   var = dereference( AX[0] );
   var1 = dereference( AX[1] );
   if(( type( var1 ) == tcon ) && ( subtype( var1 ) == tnum ))
   {  if( numvalue(var1) >15 ) { printf("Illegal Set variable %d \n",
                                         numvalue(var1));
                                printf("Set variable 0 used.\n");
                                var1 = 0;  };
      val = stuck(numvalue(var1));
      if( !unify(val,var) ) fail();  }
   else fail();
                                                       }

integer()

{  long val;

   val = dereference( AX[0] );
   if(( type(val) != tcon ) || ( subtype(val) != tnum )) fail();  }
```

```
/*  file  escape4.c  */
/*  seq   1.3.22     */

#include "regs.h"
#include "structures.h"
#include "spacevars.h"
#include "typvars.h"
#include "routines.h"

struct strng NOP_str = { 3, " NOP" };
struct strng trust_str = { 13, " trust_me_else" };
struct strng retry_str = { 13, " retry_me_else" };
struct strng try_str = { 11, " try_me_else" };
struct strng fail_str = { 4, " fail" };
extern struct strng BLANK;

asserta()

{  int a,ra,na;
   long val1,val2;

   val1 = dereference( AX[0] );
   val2 = dereference( AX[1] );
   if(( type( val1 ) != tcon ) || type( val2 ) != tcon ) fail();
   else  {
   a = locate( Ptbl, Pno, Cspace[value(val1)].inst );
   a = Ptbl[a].addr;
   na = locate( Ltbl, Lbloc, Cspace[value(val2)].inst );
   na = Ltbl[na].addr;
   ra = locate( Ltbl, Lbloc, Cspace[a].args );
   ra = Ltbl[ra].addr;
   if( !equalstr( Cspace[ra].inst, fail_str ))
      Cspace[na].args = Cspace[a].args;
   else  Cspace[a+1].inst = Cspace[value(val2)].inst;
   Cspace[a].args = Cspace[value(val2)].inst;
   if( equalstr( Cspace[ra].inst, NOP_str ))
   {  Cspace[ra].inst = trust_str;
      Cspace[ra].args = fail_str;   }
   else Cspace[ra].inst = retry_str;      };   }

assertz()

{  int a,ea,na;
   long val1,val2;

   val1 = dereference( AX[0] );
   val2 = dereference( AX[1] );
   if(( type( val1 ) != tcon ) || ( type( val2 ) != tcon )) fail();
   else  {
```

```
   a = locate( Ptbl, Pno, Cspace[value(val1)].inst );
   a = Ptbl[a].addr;
   na = locate( Ltbl, Lbloc, Cspace[value(val2)].inst );
   na = Ltbl[na].addr;
   ea = locate( Ltbl, Lbloc, Cspace[a+1].inst );
   ea = Ltbl[ea].addr;
   Cspace[a+1].inst = Cspace[value(val2)].inst;
   if( equalstr( Cspace[ea].inst, NOP_str ))
   {  Cspace[ea].inst = try_str;
      Cspace[ea].args = Cspace[a+1].inst;   }
   else
   {  Cspace[ea].inst = retry_str;
      Cspace[ea].args = Cspace[a+1].inst;   };
   Cspace[na].inst = trust_str;
   Cspace[na].args = fail_str;   };  }

retracta()

{  int a,oa,ra;
   long val;

   val = dereference( AX[0] );
   if( type(val) != tcon ) fail();
   else  {
   a = locate( Ptbl, Pno, Cspace[value(val)].inst );
   a = Ptbl[a].addr;
   oa = locate( Ltbl, Lbloc, Cspace[a].args );
   oa = Ltbl[oa].addr;
   if( equalstr( Cspace[oa].inst, fail_str )) fail();
   else if( equalstr( Cspace[oa].inst, NOP_str )) retractp();
   else
   {  ra = locate( Ltbl,Lbloc, Cspace[oa].args );
      Cspace[a].args = Ltbl[ra].name;
      ra = Ltbl[ra].addr;
      if( equalstr( Cspace[ra].inst, trust_str ))
      {  Cspace[ra].inst = NOP_str;
         Cspace[ra].args = BLANK;   }
      else Cspace[ra].inst = try_str;
                                    };  };  }

retractp()

{  int a,oa;
   long val;

   val = dereference( AX[0] );
   if( type(val) != tcon ) fail();
   else  {
   a = locate( Ptbl, Pno, Cspace[value(val)].inst );
   a = Ptbl[a].addr;
   oa = locate( Ltbl, Lbloc, Cspace[a].args );
   oa = Ltbl[oa].addr;
   if( equalstr( Cspace[oa].inst, fail_str )) fail();
   else
   {  Cspace[oa].inst = fail_str;
      Cspace[oa].args = BLANK;  };
                                };  }
```

```
/*  file  escape5.c  */
/*  seq   1.         */

#include "regs.h"
#include "structures.h"
#include "spacevars.h"
#include "typvars.h"
#include "routines.h"

rplac()

{  int temp;

   if( mode == read )
   {  temp = S;
      S = H;
      H = temp;
      while(( stack[value(H)]&tcdr )&&( stack[value(H)] != NIL))
         H = value(stack[H]);
      mode = write;
         }   }

nrplac()

{  int temp;

   if( mode == write )
   {  temp = S;
      S = H;
      H = temp;
      mode = read;
      }   }

rplacd()

{  int temp;

   if( mode == read )
   {  temp = S;
      S = H;
      H = temp;
      mode = write;   }   }

eq()
/* test for term equality */
{
if (!equiv(dereference(AX[0]), dereference(AX[1])))
   fail();
}
```

```
equiv(arg1,arg2)
/* test for term equality */
long arg1, arg2;
{
long temp;
if (type(arg1) != type(arg2))
   return(0);
switch (type(arg1)) {
   case (tvar):
      arg1 = dereference(arg1);
      arg2 = dereference(arg2);
   case (tcon):
      if (arg1 != arg2)
         return(0);
      return(1);
   case (tlst):
   case (tstr):
      if( !equiv( stuck(value(arg1)), stuck(value(arg2)) ))
         return(0);
      else {
         temp = value(arg1)+1;
         arg1 = stuck( temp );
         if(!( arg1 & tcdr ))
            arg1 = tag( temp, tlst );
         temp = value(arg2)+1;
         arg2 = stuck( temp );
         if(!( arg2 & tcdr ))
            arg2 = tag( temp, tlst );
         if( !equiv( arg1, arg2 ))
             return(0);
         else
            return(1);
         }
   }
}
```

```
/*  file  basics.c  */
/*  seq   1a.3.14    */

#include "regs.h"
#include "structures.h"
#include "spacevars.h"
#include "typvars.h"
#include "routines.h"
#include "parameters.h"
#include "instvars.h"
#include "osdefs.h"
#include "ib.h"
#include "orp.h"
#include "env.h"
#include "cp.h"
#include "msg.h"
#include "timing.h"

#define ID(x)  (x & 0x1fffffff)
extern int quit;
extern int IP;

int f1;

fail()

{
long var,i;

mixtbl[FAILS]++;
f1 = 1;
if (DBT) printf("   backtracking...\n");
if ( B == STKbase ) {
   if (cpid == 0) {
      P = 0;
      TP(5);
      }
   else {
      TP(4);
      while( TR != TRLbase ) {
         TP(4);
         var = stuck(--TR);
         if( stuck(var) & tcdr ) i = tcdr;
         else i = 0;
         stick(var, tag( var, tvar ) | i );
         }
      OS_SEND(cpid,FAIL,parent);
      }
   }
```

```
else {
   i = stuck(B-sB);
   if (cptype(i) == pseudocp)
      B = findbk(B);
   if (B == STKbase) {
      if (cpid == 0) {
         P = 0;
         TP(10);
         }
      else {
         TP(9);
         while( TR != TRLbase ) {
            TP(4);
            var = stuck(--TR);
            if( stuck(var) & tcdr ) i = tcdr;
            else i = 0;
            stick(var, tag( var, tvar ) | i );
            }
         OS_SEND(cpid,FAIL,parent);
         }
      }
   else {
      TP(5);
      next_answer(B);
      }
   }
if ( P == 0 ) {
   TP(5);
   if (cpid == 0) {
      printf("goal failure \n" );
      quit = 1;
      /*
      debug( "quit" );
      */
      }
   }
else
   TP(2);
}

next_answer(b)
long b;
{
long i,var,m,t,p,r,cc,n;
TP(3);
switch(cptype(stuck(b-sB))) {
   case(seqcp):
      TP(5);
      if (DBT) printf("   restoring from seq choice point\n");
      while( TR != stack[b-sTR] ) {
            TP(4);
            var = stuck(--TR);
            if( stuck(var) & tcdr ) i = tcdr;
            else i = 0;
            stick(var, tag( var, tvar ) | i );
            }
        i   = stuck(b-sBCE);
        E   = numvalue(i);
        E_FF = effval(i);
        CP  = stuck(b-sBCP);
        H   = stuck(b-sH);
        P   = stuck(b-sBP);
        N   = stuck(b-sN);
        W   = stuck(b-sW);
        for( i = 0; i < Rn; i++ ) AX[i] = stuck(b-i-5);
        TP(9);
        break;
    case(andcp):
        if (DBT) printf("   restoring from AND choice point\n");
        while( TR != stack[b-sTR] ) {
            TP(4);
            var = stuck(--TR);
            if( stuck(var) & tcdr ) i = tcdr;
            else i = 0;
            stick(var, tag( var, tvar ) | i );
            }
        i = stuck(b-sBCE);
        E = numvalue(i);
        E_FF = effval(i);
        P = stuck(b-sBCP);
        H = stuck(b-sH);
        N = stuck(b-sN);
        W = stuck(b-sW);
        GET_LOCK;
        p = stuck(E+P_TBL);
        m = stuck(b-sM);
        t = stuck(p+m-1);
        TP(26);
        if (t & ht) {
            TP(2);
            stick(p+m-1,t & ~ht);
            RELEASE_LOCK;
            OS_SEND(cpid,NA,ID(t));
            }
        else {
            RELEASE_LOCK;
            OS_SLEEP(cpid);
            }
        break;
    case (d_andcp):
        B = numvalue(stuck(b-sB));
        HB = stuck(b-sH);
        fail();
        return;
    case(orcp):
        TP(6);
        if (DBT) printf("   restoring from OR choice point\n");
        while( TR != stack[b-soTR] ) {
            TP(4);
            var = stuck(--TR);
            if( stuck(var) & tcdr ) i = tcdr;
            else i = 0;
            stick(var, tag( var, tvar ) | i );
            }
        i  = stuck(b-soE);
        E  = numvalue(i);
        CP = stuck(b-soCP);
        H  = stuck(b-soH);
        N  = stuck(b-soN);
        W  = stuck(b-soW);
        E_FF = effval(i);
        TP(8);
        GET_LOCK;
        cc = stuck(b-soC);
        n = stuck(b-soL);
        if ((cc == 1)||SEQ(stuck(b-ocp-cc+1))||(c0 == FAIL_FROM_OR)) {
            TP(7);
            cc++;
            while ((cc <= n) && (stuck(b-ocp-cc+1) == -1)) {
                cc++;
                TP(10);
                }
            if (cc > n) {
                TP(5);
                if (DBT) printf("   no more alternatives\n");
                B = numvalue(stuck(b-soB));
                HB = stuck(b-soH);
                RELEASE_LOCK;
                c0 = BACKWARD;
                fail();
                return;
                }
            TP(3);
            stick(b-soC, cc);
            t = stuck(b-ocp-cc+1);
            if (!SEQ(t)) {
                if (DBT) printf("   now listening to p%x\n"  ID(t))
                if (t & SUCFLAG) {
                    if (DBT) printf("   pushing bindings  continuing\n")
                    pb(t & ~SUCFLAG)
                    RELEASE_LOCK;
                    proceed();
                    }
                else {
                    RELEASE_LOCK;
                    OS_SLEEP(cpid);
                    }
                }
            else {
                for( i = 0; i < Rn; i++ ) AX[i] = stuck(b-soAR-i);
                P = t & ~SEQFLAG;
                }
            }
        else {
            TP(12);
            t = stuck(b-ocp-cc+1);
            stick(b-ocp-cc+1, t & ~SUCFLAG);
            RELEASE_LOCK;
            OS_SEND(cpid,NA,ID(t));
            OS_SLEEP(cpid);
            }
        break;
    default:
        printf("next_answer: error\n");
    }
}

int btype;

findbk(cpptr)
    long cpptr;
{
    short a = 0;
    short l;
    long w,z,y,v,r,p,x;

         if (DBT) printf("    (intelligently)...\n");
         E = numvalue(stuck(cpptr-sBCE));
         l = stuck(cpptr-sN);
         r = stuck(E+CP_TBL);
         v = stuck(E+DT_TBL);
         p = stuck(E+P_TBL);
         if ((stuck(r+l-1) != UNKNOWN)) {
             if ((stuck(E+sCE) & EXIT) || l == 1) {
                 w = numvalue(stuck(cpptr-sB));
                 while (w != STKbase && (cptype(y=stuck(w-sB))==pseudocp))
                     w = numvalue(y);
                 btype = 3;
                 killprocs(l,E,btype);
                 if (DBT) printf("   type III\n");
                return(w);
                }
            else {
                x = typeII(v,1);
                btype = 2;
                killprocs(1,E,btype);
                if (DBT) printf("   type II\n");
                mixtbl[TYPEII]++;
                }
            }
        else     /* backtracking of type I */
        {   x = typeI(v,1);
            btype = 1;
            killprocs(1,E,btype);
            if (DBT) printf("   type I\n");
            mixtbl[TYPEI]++;
        };
        while (x!=0  && (w=stuck(r+x-1)) == NONE)
            x = typeII(v,x);
        if (x == 0) {
            w = numvalue(stuck(E+sCB));
            while (w != STKbase && (cptype(y=stuck(w-sB)) == pseudocp))
                w = numvalue(y);
            }
        return(w);
    }

killprocs(lit,e,bt)
short lit;
long e;
short bt;
{
short x,i;
long bta = stuck(e+BT_TBL);
long proc, p;
struct strng list, klist();
list = klist(bta,lit,bt);
while (length(list) != 0) {
    i = delimit(list,',');
    if (i == 0) {
        x = getnum(list);
        if (x == 0)
            return;
        i = length(list);
        }
    else
        x = getnum(substr(list,1,i-1));
    list = substr(list,i+1,length(list)-i);
    p = stuck(e+P_TBL);
    proc = stuck(p+x-1);
    if (proc != -1) {
       OS_SEND(cpid,KILL,ID(proc));
       stick(p+x-1,-1);
       }
    }
}

struct strng klist(b,m,bt)
long b;
short m;
short bt;
{
switch (bt) {
   case 1:
      return(Cspace[b+(m-1)*2+1].inst);
   case 2:
   case 3:
      return(Cspace[b+(m-1)*2+1].args);
   }
}

typeI(b,m)
long b,m;
{
return(getnum(Cspace[b+(m-1)*2].inst));
}

typeII(b,m)
long b,m;
{
return(getnum(Cspace[b+(m-1)*2].args));
}
```

```
#include "regs.h"
#include "parameters.h"
#include "process.h"
#include "osdefs.h"
#include "fork.h"
#include "mem.h"

extern struct process *pt[];

long newpid()
{
short i = 0;
for (i = 0; i < MAXPROCS; i++) {
   if (!(pt[i] -> occ)) {
      pt[i] -> occ = 1;
      return(i);
      }
   }
if (i == MAXPROCS) {
   printf("newpid: process table overflow\n");
   pquit();
   }
}

extern long newpage();
newmem(p)
struct process *p;
{
long base;
base = newpage();
p -> Wbase = base;
p -> HPbase = base + WINSIZE;
p -> STKbase = base  + WINSIZE + HPSIZE;
p -> TRLbase = base  + WINSIZE + HPSIZE + STKSIZE;
p -> PDL = base + PMSIZE;
p -> W = p -> Wbase;
p -> H = p -> HPbase;
p -> S = p -> HPbase;
p -> HB = p -> HPbase;
p -> E = p -> STKbase - 1;
p -> B = p -> STKbase;
p -> TR = p -> TRLbase;
}
```

```
/*  file  get1.c  */
/*  seq   1.3.6   */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "timing.h"

get_variable( arg )
   struct strng arg;

{  int i,n,flag;
   char c;
   long val;

   flag = get_2args( arg, &c, &i, &n );
   if( ! flag ) NOP();
   else  {
      val = AX[i];
      if( c == 'Y' ) {
         TP(5);
         stick(E+envsz()+n, val);
         }
      else {
         AX[n] = val;
         TP(3);
         }
      }
}

get_value( arg )
   struct strng arg;

{  long val,var;
   int i,n,flag;
   char c;

   flag = get_2args( arg, &c, &i, &n );
   if( ! flag )
      NOP();
   else  {
      if( c == 'Y' ) {
         var = dereference( stuck(E+envsz()+n) );
         TP(6);
         }
      else {
         TP(3);
         var = dereference( AX[n] );
```

```
            }
        val = dereference( AX[i] );
        if( ! unify( val, var ))
            fail();
        else if ( c != 'Y' )
            AX[n] = val;
        }
}

get_constant( arg )
    struct strng arg;

{
long var,val;
int x,i;
struct strng C,Ai;

x = delimit( arg, ',' );
if( x == 0 )
    NOP();
else {
    C = substr( arg, 1, x-1 );
    Ai = substr( arg, x+1, length(arg)-x );
    i = get_Areg( Ai );
    if( i < 0 )
        NOP();
    else  {
        TP(1);
        var = dereference( AX[i] );
        val = get_consts( C );
        if( !unify( val, var ))
            fail();
        }
    }
}
```

```
/*  file  get2.c  */
/*  seq   1a.3.7   */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "timing.h"


get_nil( arg )
   struct strng arg;

{  int i,x;
   long var,val;

   i = get_Areg( arg );
   if( i < 0 )
      NOP();
   else  {
      TP(2);
      var = dereference( AX[i] );
      val = tag( NIL, tcon );
      if( !unify( var, val ))
         fail();
      }
}

get_structure( arg )
   struct strng arg;

{  int i,x;
   long var,val;
   struct strng F,Ai;

   x = delimit( arg, ',' );
   if( x == 0 ) NOP();  else  {
   F = substr( arg, 1, x-1 );
   Ai = substr( arg, x+1, length(arg)-x );
   i = get_Areg( Ai );
   if( i < 0 ) NOP();  else  {
   var = dereference( AX[i] );
   val = get_consts( F );
   if( type(var)  == tvar )
   {
      TP(4);
      stick(H, val);
      val = tag( H++, tstr );
      stick(value(var), val);
```

```
        trail( value(var) );
        mode = write;            }
     else if(( type(var) == tstr ) && ( stuck(value(var)) == val )) {
        TP(5);
        S = value(var)+1;
        mode = read;         }
     else {
        TP(2);
        fail();
        } } } }

get_list( arg )
     struct strng arg;

{  int i,x;
     long var,val;

     i = get_Areg( arg );
     if( i < 0 ) NOP();  else  {
     var = dereference( AX[i] );
     if( type(var) == tlst ) {
        TP(3);
        S = value(var);
        mode = read;
        }
     else if( type(var) == tvar )
     {  TP(3);
        val =tag( H, tlst );
        unify( val, var );
        mode = write;            }
     else {
        TP(2);
        fail();
        } } }
```

```
#include <stdio.h>
#include "graph.h"
#include "structures.h"

extern short ANDFLAG, ORFLAG, IBFLAG;
extern int sim_cyc;
extern FILE *gfile;

ggraph(file)
struct strng file;
{
short spec,i,j = 0;
char line[80], name[80], code[4];
int x = 1, cyc = 0, pt, procs, xdelta, ydelta;
float tprocs, interval, y, maxy;
FILE *ggf;
spec = delimit( file, '.' );
file.ch[spec+1] = 'g';
file.ch[spec+2] = 'g';
file.ch[spec+3] = 'r';
file.ch[spec+4] = 'a';
file.ch[spec+5] = 'p';
file.ch[spec+6] = 'h';
file.ch[spec+7] = '\0';
if(( ggf = fopen( &file.ch[1], "w" )) == NULL ) {
   printf("Unable to open ggraph file\n");
   exit(1);
   }
else
   printf("Writing file %s \n", file.ch );
fclose(gfile);
gfile = fopen( GRAPH_FILE, "r");
if (sim_cyc < MAXPOINTS)
   interval = 1;
else
   interval = (sim_cyc/MAXPOINTS) + 1;
fprintf(ggf,"das PROC\n");
while (fgets(line,80, gfile) != NULL) {
   sscanf(line,"%d %d",&pt,&procs);
   tprocs += procs;
   cyc++;
   if (cyc == interval) {
      y = tprocs/interval;
      fprintf(ggf,"%d   %.0f\n",x++,y);
      if (y > maxy)
         maxy = y;
      cyc = 0;
      tprocs = 0;
      }
   }
```

```
#include "osdefs.h"
#include "fork.h"
#include "msg.h"
#include "regs.h"
#include "cp.h"
#include "ib.h"
#include "orp.h"
#include "process.h"
#include "timing.h"
#include "parameters.h"
#include "mem.h"
#include "typvars.h"

struct process *pt[MAXPROCS];
extern long newpid(), *processors;

long OS_FORK(par,k,P,parB,i)
long par;
short k;
long P;
long parB;
short i;
{
struct process *parent, *child;
long pid = newpid();
short j;

GET_LOCK;
EF[PROCESSOR]++;
TP(FORK_TIME);
parent = pt[par];
child = pt[pid];
child -> P = P;
child -> pid = pid;
child -> parent = par;
child -> kind = k;
child -> parB = parB;
child -> i = i;
for (j = 0; j < Rsiz; j++)
  child -> AX[j] = parent -> AX[j];
newmem(child);
child -> N = 0;
child -> CP = 0;
child -> ictr = 0;
child -> cut = 0;
child -> E_FF = 0;
child -> c0 = FORWARD;
child -> c1 = -1;
child -> state = RUNNABLE;
newprocess(-1);
```

```
if ((dbg) || tracing(child -> pid) || tracing(child -> parent))
   printf("Created %s process, parent = %x, id = %x\n",
   kindstr(child -> kind),child -> parent,child -> pid);
RELEASE_LOCK;
return(pid);
}

OS_SEND(s, msg, d)
long s;
short msg;
long d;
{
short i, oPROCESSOR;
long e, lit, t, p, cc, n, b, c, ocpid, var;
struct process *src, *dst, *tmp;
if ((s < 0) || (s > MAXPROCS) || (d < 0) || (d > MAXPROCS)) {
   printf("OS_SEND: invalid process id: s = %x, d = %x\n",s,d);
   debug("dbg");
   exit(1);
   }
src = pt[s];
dst = pt[d];
if (DBT) printf("   p%x sending %s to p%x\n", s, msgstr(msg), d);
oPROCESSOR = PROCESSOR;
if (PROCESSOR != PROCESSORS) {
   GET_LOCK;
   EF[PROCESSOR]++;
   TP(MSG_WRITE_TIME);
   PROCESSOR = PROCESSORS;
   }
switch(msg) {
   case(FAIL):
MSG_IS_FAIL:
      OS_DIE(src -> pid);
      switch (src -> kind) {
         case(AND):
         case (DET_AND):
            dst -> c0 = FAIL_FROM_AND;
            dst -> c1 = src -> parB;
            if (dst -> state == SLEEPING) {
               dst -> state = RUNNABLE;
               if (notld(dst->pid))
                  newprocess(-1);
               else
                  dst -> state = RUNNING;
               }
            break;
         case(OR):
            if (dst -> state == RUNNING)
               GET_LOCK;
```

```
fprintf(ggf,"dae\n");
i = spec;
/* remove leading pathnames and trailing suffix */
while ((file.ch[i] != '/') && (i > 0))
   i--;
i++;
while (file.ch[i] != '.')
   name[j++] = file.ch[i++];
name[j++] = '\0';
if (x <= 10)
   xdelta = 1;
else if (x <= 20)
   xdelta = 2;
else if (x <= 50)
   xdelta = 5;
else if (x <= 100)
   xdelta = 10;
else if (x <= 200)
   xdelta = 20;
else xdelta = 50;
if (maxy <= 10)
   ydelta = 1;
else if (maxy <= 20)
   ydelta = 2;
else if (maxy <= 50)
   ydelta = 5;
else if (maxy <= 100)
   ydelta = 10;
else if (maxy <= 500)
   ydelta = 50;
else ydelta = 100;
code[0] = ' ';
code[1] = ' ';
code[2] = ' ';
code[3] = '\0';
if (ANDFLAG)
   code[0] = 'A';
if (ORFLAG)
   code[1] = 'O';
if (IBFLAG)
   code [2] = 'I';
fprintf(ggf,"xst %d %d %d %d\n",XTICKS,xdelta,XSTART,XORG);
fprintf(ggf,"yst %d %d %d %d\n",YTICKS,ydelta,YSTART,YORG);
fprintf(ggf,"xgr off\n");
fprintf(ggf,"ygr off\n");
fprintf(ggf,"xla time (scaled)\n");
fprintf(ggf,"yla runnnable processes\n");
fprintf(ggf,"gti %s (%s)\n",name,code);
fprintf(ggf,"fra off\n");
fprintf(ggf,"ssw off\n");
fprintf(ggf,"dra %s.grn\n",name);
fclose(gfile);
unlink(GRAPH_FILE);
}
```

```
/*  file ib.c */

#include "regs.h"
#include "structures.h"
#include "spacevars.h"
#include "routines.h"
#include "ib.h"
#include "env.h"
#include "cp.h"
#include "timing.h"

i_allocate(args)        /* i_alloc #lits.BT_TBL.J_TBL.N */
   struct strng args;
{
   short i;
   int x,y,z;
   struct strng arg1,arg2,arg3;
   long ce;

   x = delimit(args,',');
   if (x == 0) {
      printf("i_allocate: invalid format %s\n", args);
      exit(1);
      }
   else {
      arg1 = substr(args,1,x-1);
      args = substr(args,x+1,length(args)-x);
      x = delimit(args,',');
      arg2 = substr(args,1,x-1);
      args = substr(args,x+1,length(args)-x);
      x = delimit(args,',');
      arg3 = substr(args,1,x-1);
      args = substr(args,x+1,length(args)-x);
      x = getnum(arg1); /* get no. of entries of CP_TBL */
      y = locate(Ltbl,Lbloc,arg2);  /* get BT_TBL */
      if (y < 0) {
         printf("i_allocate: can't find BT label %s\n",arg2);
         exit(1);
         }
      z = locate(Ltbl,Lbloc,arg3); /* J_TBL */
      if (z < 0) {
         printf("i_allocate: can't find JT label %s\n",arg3);
         exit(1);
         }
      z = Ltbl[z].addr;
      ce = E;
      if (ce < B) {
         E = B;
         TP(2);
         }
```

```
        else {
           E = ce + N + envsz();
           TP(4);
           }
        ce = ce | emask(E_FF);
        ce = ce & ~(EXIT);          /* initialize the EXIT_FLAG */
        stick(E+sCE,ce);
        stick(E+sCB,B|cut);
        stick(E+sCP,CP);
        stick(E+sCN,N);
        E_FF = 1; /* set the E_FF flip-flop */
        N = getnum(args); /* record size of perm. vars in N */
        stick(E+BT_TBL,Ltbl[y].addr);  /* store ^BT_TBL */
        stick(E+CP_TBL,H);          /* store ^CP_TBL */
        H += x;
        stick(E+P_TBL,H);           /* store ^P_TBL */
        TP(16);
        for (i = 0; i < x; i++) {
           TP(4);
           stick(H++,-1);
           }
        TP(1);
        stick(E+J_TBL,H);
        while (getnum(Cspace[z].inst) != 0) {
           TP(6);
           stick(H++,getnum(Cspace[z++].inst));
           }
        }
}

i_cut(arg)
   struct strng arg;
{
   int x;

   TP(6);
   x=getnum(arg);
   cutp();
   stick(stuck(E+CP_TBL)+x-1,B);
};

i_call(args)
   struct strng args;

{  int x,y;
   struct strng proc;

      TP(3);
      call(args);
      stick(E+LB,B);    }
```

```
extern short IBFLAG;
make(arg)
   struct strng arg;
{
   long x,y;
   IBFLAG = 1;
   x = (long)getnum(arg);
   if (E < B) {
      y = B;
      TP(3);
      }
   else {
      y = E + N + envp;
      TP(4);
      }
   stick(y+pcp-sB,B|pseudocp);
   stick(y+pcp-sBCE,E);
   stick(y+pcp-sN,x);
   B = y+pcp;
   stick(stuck(E+CP_TBL)+x-1,UNKNOWN);
   TP(11);
}

enter(arg)
   struct strng arg;
{
   int x;
   long y,u,z,w;

   x = getnum(arg);
   u = stuck(E+CP_TBL) + x - 1; /* pointer to the xth entry of CP_TBL */
   if (stuck(u) == UNKNOWN)
      return;
   z = B;
   w = stuck(z-sB);
   y = stuck(E+LB);
   TP(4);
   while ((z!=y) && (cptype(w) == pseudocp)) {
         TP(9);
         z = numvalue(w);
         w = stuck(z-sB);
   }
   if (z==y) {
      TP(6);
      stick(u,NONE);     /* if no ch-point for the call */
      }
   else {
      TP(8);
      stick(u,z);
```

```
        }
}

envsz()
{
    if (E_FF == 0) return(env);
          else return(envp);
}
```

```
/*  file  index1.c  */
/*  seq   1.3.2     */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "routines.h"
#include "env.h"
#include "cp.h"
#include "orp.h"
#include "timing.h"

struct strng cutstr = { 3," cut" };
struct strng lcutstr = { 4," lcut" };
struct strng failstr = { 4," fail" };

try_me_else ( arg )
   struct strng arg;

{  long A;
   int i,x;

   TP(23);
   if( E < B ) A = B + scp; else A = E + N + scp + envsz();
   x = locate( Ltbl, Lbloc, arg );
   if( x < 0 ) NOP();
   else  {
   for( i = 0; i < Rn; i++ ) stick(A-i-5, AX[i]);
   stick(A-sBCE, E|emask(E_FF));
   stick(A-sBCP, CP);
   stick(A-sB  , B);
   stick(A-sBP , Ltbl[x].addr);
   stick(A-sTR , TR);
   stick(A-sH  , H);
   stick(A-sN  , N);
   stick(A-sW  , W);
   cut = cutm;
   HB = H;
   B = A;   }      }

retry_me_else ( arg )
   struct strng arg;

{  int x;

   x = locate( Ltbl, Lbloc, arg );
   if( x < 0 ) NOP();
   else {
      TP(2);
```

```
        stick(B-sBP, Ltbl[x].addr);
        cut = cutm;    };   }

trust_me_else ( arg )
    struct strng arg;

{

TP(9);
if( !equalstr( arg, cutstr ) &&  !equalstr( arg, lcutstr ) &&
!equalstr( arg, failstr )) {
    printf("trust_me_else: bad arg %s\n", arg);
    exit(1);
    }

if( equalstr( arg, cutstr ) ||  equalstr( arg, lcutstr ))
    while( B != numvalue(stuck(E+sCB))) {
        kp(B);
        HB = HBval(B);
        B = numvalue(stuck(B-sB));
        }
if( !equalstr( arg, lcutstr )) {
    kp(B);
    HB = HBval(B);
    B  = numvalue(stuck(B-sB));
    }
cut = 0;
}


cutp()

{  struct strng arg;
   TP(5);
   if( stuck(E+sCB) & cutm )
       arg = cutstr;
   else arg = lcutstr;
   trust_me_else( arg );    }


cutd( arg )
   struct strng arg;

{  long x;
   TP(10);
   x = locate(Ltbl, Lbloc, arg );
   if( x < 0 ) NOP();
   x = Ltbl[x].addr;
   while( x != BPval(B) ) {
       kp(B);
```

```
          B   = numvalue(stuck(B-sB));
          }
     kp(B);
     HB = HBval(B);
     B  = numvalue(stuck(B-sB));    }

BPval(b)
long b;
{
switch(cptype(stuck(b-sB))) {
     case orcp:
          return(stuck(b-soBP));
     case seqcp:
     case andcp:
          return(stuck(b-sBP));
     }
}

HBval(b)
long b;
{
switch(cptype(stuck(b-sB))) {
     case orcp:
          return(stuck(b-soH));
     case seqcp:
     case andcp:
          return(stuck(b-sH));
     }
}
```

```
/*  file  index2.c  */
/*  seq   1.3.3     */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "routines.h"
#include "cp.h"
#include "timing.h"

try ( arg )
   struct strng arg;

{  long A;
   int i,x;

   TP(23);
   if( E < B ) A = B; else A = E + N + envsz();
   A = A + scp;
   x = locate( Ltbl, Lbloc, arg );
   if( x < 0 ) NOP();
   else  {
   for( i = 0; i < Rn; i++ ) stick(A-i-5, AX[i]);
   stick(A-sBCE, E|emask(E_FF));
   stick(A-sBCP, CP);
   stick(A-sB  , B);
   stick(A-sBP , P);
   stick(A-sTR , TR);
   stick(A-sH  , H);
   stick(A-sN  , N);
   stick(A-sW  , W);
   cut = cutm;
   HB = H;
   B = A;
   P = Ltbl[x].addr;   }      }

retry ( arg )
   struct strng arg;

{  int x;

   TP(5);
   x = locate( Ltbl, Lbloc, arg );
   if( x < 0 ) NOP();
   else   {
      stick(B-sBP, P);
      cut = cutm;
      P = Ltbl[x].addr;   }   }
```

```
trust ( arg )
   struct strng arg;

{  int x;

   TP(9);
   x = locate( Ltbl, Lbloc, arg );
   if( x < 0 ) NOP();
   else  {
      cut = 0;
      HB = HBval(B);
      B  = numvalue(stuck(B-sB));
      P  = Ltbl[x].addr;    }   }
```

```
/*  file  index3.c   */
/*  seq  1a.3.4      */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "timing.h"

switch_on_term ( args )
   struct strng args;

{  int x;

   TP(5);
   switch( type( dereference( AX[0] )))
   {
      case tstr: x = delimit( args, ',' );
                 if( x == 0 ) break;
                 args = substr( args, x + 1, length(args)-x );
      case tlst: x = delimit( args, ',' );
                 if( x == 0 ) break;
                 args = substr( args, x + 1, length(args)-x );
      case tcon: x = delimit( args, ',' );
                 if( x == 0 ) x = length(args) + 1;
                 args = substr( args, 1, x-1 );
                 break;
      case tvar: x = 0;
                                                                   };
   if( x != 0 )
   {
   x = locate( Ltbl, Lbloc, args );
   if( x <= 0 ) fail();  else  {
   P = Ltbl[x].addr;        };  };                                  }
```

```
/*  file  index4.c   */
/*  seq   1.3.5      */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "spacevars.h"
#include "routines.h"
#include "timing.h"

struct strng cdrstr = { 4, " tcdr" };

switch_on_constant( args )
   struct strng args;

{  int x,i,num;
   long val;

   TP(5);
   x = delimit( args, ',' );
   if( x == 0 ) x = length(args) + 1;
   num = getnum( substr( args, 1, x-1 ));
   args = substr( args, x+1, length(args)-x );
   x = locate( Ltbl, Lbloc, args);
   x = Ltbl[x].addr;
   val = dereference( AX[0] );
   if( type( val ) == tcon )
   {  if( subtype( val ) == tnum ) args = makenum( numvalue( val ));
      else args = Cspace[value(val)].inst;
      for( i = 0; i < num; i++)
         if( equalstr( args, Cspace[x+2*i].inst )) i = i + num+1;
      if( i == num ) fail();
      else  {
         i = i - num-2;
         x = locate( Ltbl, Lbloc, Cspace[x+2*i+1].inst );
         if( x==0 ) NOP();
         else P = Ltbl[x].addr;   };   }
   else fail();                                          }


switch_on_structure( args )
   struct strng args;

{  int x,i,num;
   long val;

   x = delimit( args, ',' );
   if( x == 0 ) x = length(args) + 1;
   num = getnum( substr( args, 1, x-1 ));
   args = substr( args, x+1, length(args)-x );
   x = locate( Ltbl, Lbloc, args);
   x = Ltbl[x].addr;
   val = dereference( AX[0] );
   TP(8);
   if(( type( val ) == tstr ) && ( type(stack[value(val)]) == tcon ) &&
      ( subtype(stack[value(val)]) != tnum ))
   {  args = Cspace[value(stack[value(val)])].inst;
      for( i = 0; i < num; i++)
         if( equalstr( args, Cspace[x+2*i].inst )) i = i+num+1;
      if( i == num ) fail();
      else  {
         i = i - num-2;
         x = locate( Ltbl, Lbloc, Cspace[x+2*i+1].inst );
         if( x==0 ) NOP();
         else P = Ltbl[x].addr;   };   }
   else fail();                                        }
```

```
#include <stdio.h>
#include "osdefs.h"
#include "process.h"
#include "routines.h"
#include "parameters.h"
#include "typvars.h"
#include "fork.h"
#include "msg.h"
#include "regs.h"
#include "timing.h"

extern struct process *pt[];

extern long *stack;
extern long *processors;
extern int tr[];

init()
{
short i;
storage();
toplevel();
processors[0] = 0;
for (i = 1; i < PROCESSORS; i++)
   processors[i] = -1;
for (i = 0; i <= PROCESSORS; i++) {
   TIME[i] = 0;
   EF[i] = 0;
   RDS[i] = 0;
   WTS[i] = 0;
   CSEC[i] = 0;
   }
for (i = 0; i < TPROCS; i++)
   tr[i] = -1;

}

storage()
{
short i,
if ((stack = (long *) calloc(Dsiz, sizeof(long))) == NULL) {
   printf("Not enough memory for data space\n");
   exit(1),
   }
for (i = 0; i < MAXPROCS; i++) {
   if ((pt[i] = (struct process *)
   calloc(1, sizeof(struct process))) == NULL) {
      printf("Not enough memory for process table\n");
      exit(1);
      }
   }
processors = (long *) calloc(PROCESSORS, sizeof(long));
TIME = (float *) calloc(PROCESSORS, sizeof(float));
EF = (float *) calloc(PROCESSORS+1, sizeof(float));
RDS = (float *) calloc(PROCESSORS+1, sizeof(float));
WTS = (float *) calloc(PROCESSORS+1, sizeof(float));
CSEC = (float *) calloc(PROCESSORS+1, sizeof(float));
}

toplevel()
{
short i;
struct process *p = pt[0];
p -> state = RUNNING;
p -> c0 = FORWARD;
p -> c1 = -1;
p -> Wbase = 0;
p -> HPbase = THPbase;
p -> STKbase = TSTKbase;
p -> TRLbase = TTRLbase;
p -> PDL = TPSIZE;
p -> occ = 1;
p -> parent = -1;
p -> pid = 0;
p -> P = 0;
p -> CP = 0;
p -> E = p -> STKbase - 1;
p -> B = p -> STKbase;
p -> TR = p -> TRLbase;
p -> H = p -> HPbase;
p -> HB = p -> HPbase;
p -> S = p -> HPbase;
p -> N = 0;
p -> W = 0;
p -> E_FF = 0;
p -> cut = 0;
for( i = 0; i < Rsiz; i++ )
   p -> AX[i] = tag( -1, tvar );
p -> ictr = 0;
}
```

```
/*  file  instrmt.c  */
/*  seq   1.2.1      */

#include <stdio.h>
#include "structures.h"
#include "instvars.h"
#include "parameters.h"
#include "osdefs.h"
#include "regs.h"
#include "timing.h"

int mixtbl[MIXSZ], inst_total;

extern int ANDFLAG, ORFLAG, IBFLAG, scheck;
extern struct tbl decode_tbl[];
extern struct strng name;
extern struct strng BLANK;


initmix()

{  int i;

   inst_total = 0;
   name = BLANK;
   for( i = 0; i < MIXSZ; i++) mixtbl[i] = 0;
   mixtbl[MXPDL] = Dsiz;
                                                    }


short Tr, Tw, Ts, Tef;
extern float avg;

prntmix( file )
   struct strng file;

{  int i,spec,maxproc;
   int j,k,l,m;
   FILE *fp, *fopen();
   float percent;
   double exinfo[2],tmp;
   char *cstring;
   float tTIME = 0, tRDS = 0, tWTS = 0, tEF = 0, tCSEC = 0;

   spec = delimit( file, '.' );
   file.ch[spec+1] = 'd';
   file.ch[spec+2] = 'a';
   file.ch[spec+3] = 't';
   file.ch[spec+4] = 'a';
```

```
    file.ch[spec+5] = '\0';
    switch(ANDFLAG + ORFLAG + IBFLAG) {
       case 0: cstring = "sequential execution";
               break;
       case 1: if (ANDFLAG)
                  cstring = "AND-parallelism";
               else if (ORFLAG)
                  cstring = "OR-parallelism";
               else
                  cstring = "intelligent backtracking";
               break;
       case 2: if (!ANDFLAG)
                  cstring = "OR-parallelism, intelligent backtracking";
               else if (!ORFLAG)
                  cstring = "AND-parallelism, intelligent backtracking";
               else
                  cstring = "AND/OR parallelism";
               break;
       case 3: cstring = "AND/OR-parallelism, intelligent backtracking";
               break;

       }
    if(( fp = fopen( &file.ch[1], "w" )) == NULL ) fp = stdout;
    else printf("Writing file %s \n", file.ch );
    fprintf( fp, "\n %s \n", file.ch );
    fprintf( fp, "Parallel version, %d processors\n", PROCESSORS);
    fprintf( fp, "Compiled for %s\n", cstring);
    if (scheck == 1)
       printf("   (System saturation not permitted)\n");
    for( i = 0; i < NUMINSTS; i++ ) {
       percent = ( mixtbl[i] * 100.0 ) / inst_total;
       fprintf( fp,"%30s \t %d \t %5.2f \n",
       decode_tbl[i].name.ch, mixtbl[i], percent );
       }
    fprintf( fp, "\n%30s \t %d \n \n", "TOTAL", inst_total );
    fprintf( fp, "%30s \t %d \n", "failures", mixtbl[FAILS] );
    fprintf( fp, "%30s \t %d \n", "type I", mixtbl[TYPEI] );
    fprintf( fp, "%30s \t %d \n", "type II", mixtbl[TYPEII] );
    fprintf( fp, "%30s \t %d \n", "unifications", mixtbl[UNIFS] );
    fprintf( fp, "%30s \t %d \n", "unify routine", mixtbl[UNIFR] );
    fprintf( fp, "%30s \t %d \n", "bindings", mixtbl[BINDS] );
    fprintf( fp, "%30s \t %d \n", "escapes", mixtbl[ESCPS] );
    fprintf( fp, "%30s \t %d \n", "dereferences", mixtbl[DERFS] );
    fprintf( fp, "%30s \t %d \n", "binding trails", mixtbl[TRALS] );
    j=mixtbl[MXTRL];
    k=mixtbl[MXSTK];
    l=mixtbl[MXHEP];
    m=mixtbl[MXWIN];
    fprintf( fp, "%30s \t %d \n", "maximum trail", j );
    fprintf( fp, "%30s \t %d \n", "maximum stack", k );
    fprintf( fp, "%30s \t %d \n", "maximum heap", l );
    fprintf( fp, "%30s \t %d \n", "maximum window", m );
    fprintf( fp, "%30s \t %d \n", "memory  reads", mixtbl[READS] );
    fprintf( fp, "%30s \t %d \n", "memory writes", mixtbl[WRITS] );
    fprintf( fp, "%30s \t %d \n", "context swaps ", mixtbl[SWAPS]);
    fprintf( fp, "\n\n");
    fprintf( fp, "\t\tPROCESSOR STATISTICS\n\n");
    fprintf( fp,
    "PROC\tuCYCLES\t  READS\t WRITES\t    EF\t   CSEC\n\n");
    EF[PROCESSORS] = 0;
    TIME[PROCESSORS] = 0;
    for (i = 0; i <= PROCESSORS; i++) {
       if (i == PROCESSORS)
          fprintf(fp, "\nHOST");
       else
          fprintf(fp, "%d", i);
       fprintf(fp, "\t%.0f\t%.0f\t%.0f\t%.0f\t%.0f\n",
       TIME[i], RDS[i], WTS[i], EF[i], CSEC[i]);
       tTIME += TIME[i];
       tRDS += RDS[i];
       tWTS += WTS[i];
       tEF += EF[i];
       tCSEC += CSEC[i];
       }
    fprintf(fp, "\n");
    fprintf(fp, "TOTAL\t%.0f\t%.0f\t%.0f\t%.0f\t%.0f\n",
    tTIME,tRDS,tWTS,tEF,tCSEC);
    Tr = 1; Tw = 1; Ts = 2; Tef = 100;
    extime(exinfo);
    maxproc = exinfo[1];
    fprintf(fp, "\nEXECUTION TIME: %.0f cycles, processor %d\n",
    exinfo[0], maxproc);
    fprintf(fp, "\nunder the following assumptions:\n");
    fprintf(fp, "Tr = %d, Tw = %d, Ts = %d, Tef = %d\n",
    Tr, Tw, Ts, Tef);
    tmp = TIME[maxproc] + CSEC[maxproc] + EF[maxproc];
    fprintf(fp, "\nExecution time weights:\n");
    fprintf(fp, "internal        %.3f\n",
    (TIME[maxproc]-RDS[maxproc]-WTS[maxproc])/tmp);
    fprintf(fp, "reads           %.3f\n",RDS[maxproc]/tmp);
    fprintf(fp, "writes          %.3f\n",WTS[maxproc]/tmp);
    fprintf(fp, "sync            %.3f\n",CSEC[maxproc]/tmp);
    fprintf(fp, "os              %.3f\n",EF[maxproc]/tmp);
    fprintf(fp, "\nsync accesses = %.0f\n",2*tCSEC);
    fprintf(fp, "\nsync %% = %.0f\n",(2*tCSEC/(2*tCSEC+tRDS+tWTS))*100);
    fprintf(fp, "avg # runnable processes = %.0f\n",avg);
    }

extime(info)
double *info;
```

```
{
double tmp = 0;
double max = 0;
short maxi;
short i;
for (i = 0; i < PROCESSORS; i++) {
   tmp = TIME[i] - RDS[i] - WTS[i] + RDS[i]*Tr + WTS[i]*Tw +
   CSEC[i]*Ts + EF[i]*Tef;
   if (tmp > max) {
      max = tmp;
      maxi = i;
      }
   }
info[0] = max;
info[1] = maxi;
}
```

```
/*  file load.c     */
/*  seq   1.2.2     */

#include <stdio.h>
#include "structures.h"
#include "spacevars.h"
#include "parameters.h"
#include "routines.h"

struct strng pstring = { 9, " procedure" };
struct strng endstr  = { 3, " end" };
struct strng flstr   = { 4, " fail" };
struct strng headstr = { 20, " # Simulator Data Run" };

extern struct strng name;
extern struct strng BLANK;

load( f_name )
   struct strng f_name;

{  int x,y,ch;
   struct strng inputstr();
   struct strng line;
   FILE *fp, *ftemp, *fopen();

   ch = 1;
   x = delimit( f_name, '.' );
   if( length( f_name ) == 0 ) fp = stdin;
   else if(( x == 0 ) || ( getch( f_name, x+1 ) != 'w' ))
   {  printf("Program file should have .w specifier \n");
      return( 0 );   }
   else if(( fp = fopen( &f_name.ch[1], "r" )) == NULL )
   {  printf("File %s unavailable \n", f_name.ch );
      return( 0 );  }
   while(( ch != 0 ) && ( Pno != Psiz ) && ( Lbloc != Lsiz ))
   {  line = inputstr( fp );
      if( equalstr( line, endstr )) ch = 0;
      else if( getch( line, 1 ) == '!' )
      {  line = substr( line, 2, length(line)-1);
         if( length( name ) == 0 ) name = headstr;
         load( line );   }
      else if(( getch( line, 1 ) == '#' ) && ( length(name) == 0 ))
         name = line;
      else if( getch( line, 1 ) != '#' )
      {  x = delimit( line, ' ' );
         if(( x != 0 ) && equalstr( substr( line, 1, x-1 ), pstring ))
         {  Pno++;
            Ptbl[Pno].name = substr( line, x+1, length(line) - x );
            Ptbl[Pno].addr = Caddr;                                }
         else if( length(line) != 0 )  {
            y = delimit( line, ':' );
            if( y != 0 ) {
               Lbloc++;
               Ltbl[Lbloc].name = substr( line, 1, y-1);
               Ltbl[Lbloc].addr = Caddr;
               if( x <= y + 1 ) y++;
               line = substr( line, y + 1, length(line)-y );   };
            if( length(line) > 0 ) {
            x = delimit( line, ' ' );
            if( x == 0 )
            {  Cspace[Caddr].inst = line;
               Cspace[Caddr++].args = BLANK;  }
            else  {
               Cspace[Caddr].inst = substr( line, 1, x-1 );
               Cspace[Caddr++].args = substr( line, x+1, length(line)-x ); }}}
   };  };
   fclose( fp );
   if( ch == 0 )
   {  printf("pau load %s \n", f_name.ch );
      printf("Cspace = %d \n",Caddr-1);
      if( length( name ) == 0 ) name = headstr;
      return( 1 );   }
   else if( Pno == Psiz )
   {  printf("Procedure Symbol Table overflow, load terminated. \n" );
      return( 0 );     }
   else
   {  printf("Statement Symbol Table overflow, load terminated. \n" );
      return( 0 );     }    }
```

```
#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "orp.h"
#include "routines.h"
#include "msg.h"
#include "cp.h"
#include "timing.h"

extern short ORFLAG;
try_p(args)
struct strng args;
{
short x,i;
int n;
long l, A;
ORFLAG = 1;
x = delimit( args, ',' );
if( x == 0 ) {
   printf("try_p: bad args %s\n", args);
   exit(1);
   }
n = getnum(substr( args, 1, x-1 ));
l = locate(Ltbl, Lbloc, substr( args, x+1, length(args)-x ));
if ((n <= 1) || (l < 0)) {
   printf("try_p: unable to locate label %s\n", args);
   exit(1);
   }
if( E < B ) {
   TP(2);
   A = B;
   }
else {
   TP(4);
   A = E + N + envsz();
   }
A = A + ocp + n;
stick(A - soB, B | orcp);
stick(A - soN, N);
stick(A - soE, E | emask(E_FF));
stick(A - soCP, CP);
stick(A - soBP,Ltbl[l].addr);
stick(A - soH, H);
stick(A - soTR, TR);
stick(A - soL, n);
stick(A - soC, 1);
stick(A - soW, W);
for (i = 0; i < Rn; i++)
   stick(A-soAR-i,AX[i]);
HB = H;
```

```
cut = cutm;
B = A;
TP(22);
for (i = 2; i <= n; i++) {
   TP(3);
   stick(A-ocp-i+1,-1);
   }
GET_LOCK;       /* must lock choice point until completely built */
}

retry_p(args)
struct strng args;
{
short x;
short n;
long l, ch;
x = delimit( args, ',' );
if( x == 0 ) {
   printf("retry_p: bad args %s\n", args);
   exit(1);
   }
n = getnum(substr( args, 1, x-1 ));
l = locate(Ltbl, Lbloc, substr( args, x+1, length(args)-x ));
if ((n <= 1) || (l <= 0)) {
   printf("retry_p: unable to locate label %s\n", args);
   exit(1);
   }

TP(4);
switch(scheck) {
   case (0):
      ch = OS_FORK(cpid,OR,Ltbl[l].addr,B,n);
      stick(B-ocp-n+1, ch);
      break;
   case (1):
      if (!SAT) {
         ch = OS_FORK(cpid,OR,Ltbl[l].addr,B,n);
         stick(B-ocp-n+1, ch);
         }
      else
         stick(B-ocp-n+1,Ltbl[l].addr | SEQFLAG);
      break;
   }
}

trust_p(args)
struct strng args;
{
short x;
short n;
```

```
long l, ch;
x = delimit( args, ',' );
if( x == 0 ) {
   printf("trust_p: bad args %s\n", args);
   exit(1);
   }
n = getnum(substr( args, 1, x-1 ));
l = locate(Ltbl, Lbloc, substr( args, x+1, length(args)-x ));
if ((n <= 1) || (l <= 0)) {
   printf("trust_p: unable to locate label %s\n", args);
   exit(1);
   }
TP(7);
switch (scheck) {
   case 0:
      ch = OS_FORK(cpid,OR,Ltbl[l].addr,B,n);
      stick(B-ocp-n+1, ch);
      break;
   case 1:
      if (!SAT) {
         ch = OS_FORK(cpid,OR,Ltbl[l].addr,B,n);
         stick(B-ocp-n+1, ch);
         }
      else
         stick(B-ocp-n+1,Ltbl[l].addr | SEQFLAG);
      break;
   }
P = stuck(B-soBP);
RELEASE_LOCK;
}

cut_p()
{
cutp();
OS_SEND(cpid,CUT,parent);
}

try_me_else_p(args)
struct strng args;
{
short x,i;
int n;
long l, A;
ORFLAG = 1;
x = delimit( args, ',' );
if( x == 0 ) {
   printf("try_p: bad args %s\n", args);
   exit(1);
   }
n = getnum(substr( args, 1, x-1 ));
```

```
l = locate(Ltbl, Lbloc, substr( args, x+1, length(args)-x ));
if ((n <= 1) || (l < 0)) {
   printf("try_me_else_p: unable to locate label %s\n", args);
   exit(1);
   }
if( E < B ) {
   TP(2);
   A = B;
   }
else {
   TP(4);
   A = E + N + envsz();
   }
A = A + ocp + n;
stick(A - soB, B | orcp);
stick(A - soN, N);
stick(A - soE, E | emask(E_FF));
stick(A - soCP, CP);
stick(A - soBP,P);
stick(A - soH, H);
stick(A - soTR, TR);
stick(A - soL, n);
stick(A - soC, 1);
stick(A - soW, W);
for (i = 0; i < Rn; i++)
   stick(A-soAR-i,AX[i]);
HB = H;
cut = cutm;
B = A;
TP(22);
for (i = 2; i <= n; i++) {
   TP(3);
   stick(A-ocp-i+1,-1);
   }
P = Ltbl[1].addr;
GET_LOCK;
}

retry_me_else_p(args)
struct strng args;
{
short x;
short n;
long l, ch;
x = delimit( args, ',' );
if( x == 0 ) {
   printf("retry_p: bad args %s\n", args);
   exit(1);
   }
n = getnum(substr( args, 1, x-1 ));
```

```
l = locate(Ltbl, Lbloc, substr( args, x+1, length(args)-x ));
if ((n <= 1) || (l <= 0)) {
   printf("retry_p: unable to locate label %s\n", args);
   exit(1);
   }

TP(4);
switch (scheck) {
   case 0:
      ch = OS_FORK(cpid,OR,P,B,n);
      stick(B-ocp-n+1, ch);
      break;
   case 1:
      if (!SAT) {
         ch = OS_FORK(cpid,OR,P,B,n);
         stick(B-ocp-n+1, ch);
         }
      else
         stick(B-ocp-n+1, P | SEQFLAG);
      break;
   }
P = Ltbl[l].addr;
}

trust_me_else_p(args)
struct strng args;
{
short x;
short n;
long l, ch;
n = getnum(args);
if (n <= 1) {
   printf("retry_p: bad arg %s\n", args);
   exit(1);
   }
TP(7);
switch (scheck) {
   case 0:
         ch = OS_FORK(cpid,OR,P,B,n);
         stick(B-ocp-n+1, ch);
         break;
   case 1:
      if (!SAT) {
         ch = OS_FORK(cpid,OR,P,B,n);
         stick(B-ocp-n+1, ch);
         }
      else
         stick(B-ocp-n+1, P | SEQFLAG);
      break;
   }
```

```
P = stuck(B - soBP);
RELEASE_LOCK;
}
```

```
            stick(src->parB-ocp-src->i+1,-1);
            if (dst -> state == RUNNING)
               RELEASE_LOCK;
            if (src -> i == stuck(src -> parB - soC)) {
               if (dst -> B == dst -> STKbase) {
                  src = dst;
                  dst = pt[dst -> parent];
                  goto MSG_IS_FAIL;
                  }
               dst -> c0 = FAIL_FROM_OR;
               dst -> c1 = src -> parB;
               dst -> state = RUNNABLE;
               if (notld(dst->pid))
                  newprocess(-1);
               else
                  dst -> state = RUNNING;
               }
           break;
        }
      break;
   case(SUC):
MSG_IS_SUC:
      switch (src -> kind) {
         case(DET_AND):
            e = numvalue(stuck(src -> parB - sBCE));
            if (dst -> state == RUNNING)
               GET_LOCK;
            t = decjte(e, src -> i);
            if (dst -> state == RUNNING)
               RELEASE_LOCK;
            if ((t == 0) && (dst -> state == SLEEPING) &&
               (listening(dst,src))) {
               dst -> state = RUNNABLE;
               if (notld(dst->pid))
                  newprocess(-1);
               else
                  dst -> state = RUNNING;
               }
            /* copy heap here */
            OS_DIE(src -> pid);
            break;
         case(AND):
            e = numvalue(stuck(src -> parB - sBCE));
            if (dst -> state == RUNNING)
               GET_LOCK;
            t = decjte(e, src -> i);
            if (dst -> state == RUNNING)
               RELEASE_LOCK;
            if ((t == 0) && (dst -> state == SLEEPING) &&
               (listening(dst,src))) {
                dst -> state = RUNNABLE;
                if (notld(dst->pid))
                   newprocess(-1);
                else
                   dst -> state = RUNNING;
                }
             lit = stuck(src -> parB - sM);
             p = stuck(e+P_TBL);
             t = stuck(p+lit-1);
             if (~(t & soll)) {
                stick(p+lit-1, t | soll | ht);
                c = stuck(e+CP_TBL);
                stick(c+lit-1, src -> parB);
                }
             else
                stick(p+lit-1, t | ht);
             break;
          case(OR):
             if (dst -> state == RUNNING)
                GET_LOCK;
             t = src -> parB - ocp - src -> i + 1;
             stick(t, stuck(t) | SUCFLAG);
             if (dst -> state == RUNNING)
                RELEASE_LOCK;
             if (src -> i == stuck(src -> parB - soC)) {
                ocpid = cpid;
                save(cpid);
                config_sim(dst -> pid);
                pb(src -> pid);
                save(dst -> pid);
                config_sim(ocpid);
                dst -> P = dst -> CP;
                if (dst -> P == 0) {
                   if (dst -> pid == 0) {
                      printf("Top level query success\n");
                      pquit();
                      }
                   else {
                      src = pt[dst -> pid];
                      dst = pt[dst -> parent];
                      goto MSG_IS_SUC;
                      }
                   }
                else {
                   dst -> state = RUNNABLE;
                   if (notld(dst -> pid))
                      newprocess(-1);
                   else
                      dst -> state = RUNNING;
                   }
                }
            break;
            }
        break;
    case(NA):
MSG_IS_NA:
        switch (dst -> kind) {
            case(AND):
                e = numvalue(stuck(dst -> parB - sBCE));
                incjte(e, dst -> i);
                dst -> cO = BACKWARD;
                dst -> state = RUNNABLE;
                if (notld(dst->pid))
                    newprocess(-1);
                else
                    dst -> state = RUNNING;
                break;
            case(OR):
                dst -> cO = BACKWARD;
                dst -> state = RUNNABLE;
                if (notld(dst->pid))
                    newprocess(-1);
                else
                    dst -> state = RUNNING;
                break;
            default:
                printf("OS_SEND: error\n");
            }
        break;
    case(KILL):
        if (dst -> kind == AND) {
            e = numvalue(stuck(dst -> parB - sBCE));
            p = stuck(e+P_TBL);
            lit = stuck(dst -> parB - sM);
            if (stuck(p+lit-1) & ht)
                incjte(e, dst -> i);
            }
        while( dst -> TR != dst -> TRLbase ) {
            TP(4);
            var = stuck(--(dst->TR));
            if( stuck(var) & tcdr ) i = tcdr;
            else i = 0;
            stick(var, tag( var, tvar ) | i );
            }
        if (dst -> B == dst -> STKbase)
             OS_DIE(dst -> pid);
        else
             dst -> cO = KILL_DESC_AND_DIE;
        break;
    case(CUT):
        if (dst -> state == RUNNING)
            GET_LOCK;
        n = stuck(src -> parB - soL);
        i = src -> i + 1;
        while (i <= n) {
            t = stuck((src->parB) - ocp - i + 1);
            if (t != -1) {
                OS_SEND(dst->pid,KILL,ID(t));
                stick((src->parB) - ocp - i + 1, -1);
                }
            i++;
            }
        if (dst -> state == RUNNING)
            RELEASE_LOCK;
        break;
    }
RELEASE_LOCK;
PROCESSOR = oPROCESSOR;
}


OS_SLEEP(pid)
long pid;
{
if ((dbg) || tracing(pid))
    printf("   putting p%x to sleep\n",pid);
pt[pid] -> state = SLEEPING;
GET_LOCK;
EF[PROCESSOR]++;
TP(MSG_WRITE_TIME + PROC_STATE_DUMP_TIME);
newprocess(PROCESSOR);
RELEASE_LOCK;
}

OS_TIMEOUT(pid)
long pid;
{
}

long OS_GETWBASE(d)
long d;
{
GET_LOCK;
EF[PROCESSOR]++;
TP(8);
RELEASE_LOCK;
return(pt[d] -> Wbase);
}

/* don't charge for EF area access, since really part of OS_GETWBASE */
```

```
long OS_GETW(d)
long d;
{
return(pt[d] -> W);
}


extern long mem[];
OS_DIE(pid)
long pid;
{
short i;
pt[pid] -> occ = 0;
mem[(pt[pid] -> HPbase - TPSIZE)/PMSIZE] = 0;
for (i = 0; i < PROCESSORS; i++) {
   if (processors[i] == pid) {
      processors[i] = -1;
      newprocess(i);
      }
   }
}
```

```
#include "parameters.h"
#include "process.h"
#include "osdefs.h"
#include "msg.h"
#include "req.h"
#include "regs.h"
#include "instvars.h"
#include "spacevars.h"
#include "mem.h"
#include "cp.h"
#include "ib.h"
#include "timing.h"
#include "structures.h"

extern long *processors;

#define PAGES ((Dsiz-TPSIZE)/PMSIZE)

long mem[PAGES];
struct process *pt[];

newprocess(proc)
short proc;
{
short i,k;
long rp[MAXPROCS], pid;
k = runnable_processes(rp);
if (k == 0)
   return;
else
   pid = scheduler(rp);

for (i = 0; i < PROCESSORS; i++)
   if (processors[i] == -1)
      break;
if (i != PROCESSORS)
   assign(pid,i);
else switch (proc) {
   case -1:
      for (i = 0; i < PROCESSORS; i++)
         if (pt[processors[i]] -> state == SLEEPING)
            break;
      if (i != PROCESSORS) {
         mixtbl[SWAPS]++;
         assign(pid,i);
         }
      break;
   default:
      mixtbl[SWAPS]++;
      assign(pid,proc);
```

```
    }
}

assign(pid,proc)
long pid;
short proc;
{
processors[proc] = pid;
pt[pid] -> state = RUNNING;
TIME[proc] += PROC_STATE_LOAD_TIME;
}

/* rp is an array of runnable process id's.  Can implement
any scheduling policy here */
scheduler(rp)
long *rp;
{
return(rp[0]);
}

runnable_processes(r)
long *r;
{
long i;
short k = 0;
for (i = 0; i < MAXPROCS; i++)
    if ((pt[i] -> occ) && (pt[i] -> state == RUNNABLE))
        r[k++] = i;
return(k);
}

running_processes()
{
short i;
short n = 0;
for (i = 0; i < PROCESSORS; i++)
    if (pt[processors[i]] -> state == RUNNING)
        n++;
return(n);
}

newpage()
{
short i;
for (i = 0; i < PAGES; i++) {
    if (!(mem[i])) {
        mem[i] = 1;
        return(TPSIZE + i*PMSIZE);
        }
    }
```

```
printf("newpage: no more memory\n");
debug("quit");
}

decjte(e,n)
long e;
short n;
{
long r,jt;
jt = stuck(e+J_TBL);
r = stuck(jt+n-1);
if (r == 0) printf("decjte: error\n");
stick(jt+n-1,r-1);
return(r-1);
}

incjte(e,n)
long e;
short n;
{
long r,jt;
TP(8);
jt = stuck(e+J_TBL);
r = stuck(jt+n-1);
stick(jt+n-1,r+1);
}

long jte(e,n)
long e;
short n;
{
TP(6);
return(stuck(stuck(e+J_TBL)+n-1));
}

extern struct tbl decode_tbl[];
struct strng waitstr = { 4, " wait" };

listening(dst,src)
struct process *dst, *src;
{
long p,j,e;
if (!equalstr(Cspace[dst->P-1].inst, waitstr))
   return(0);
e = numvalue(stuck(src -> parB - sBCE));
p = stuck(e+P_TBL);
j = stuck(e+J_TBL);
while (p < j) {
   if ((stuck(p) & ~soll ) == src -> pid)
      return(1);
   p++;
   }
return(0);
}

notld(x)
int x;
{
short i;
for (i = 0; i < PROCESSORS; i++) {
   if (processors[i] == x)
      return(0);
   }
return(1);
}
```

```
/*  file  plm.c  */
/*  seq   1a.2.0  */

#include <stdio.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/resource.h>
#include "parameters.h"
#include "structures.h"
#include "typvars.h"
#include "instvars.h"
#include "graph.h"

struct strng BLANK = { 0, " " };

struct rusage buffer;

long  Caddr = 0,
      Pno = 0,
      Lbloc = 0,
      Cno = -1;
struct tbl Ptbl[Psiz], Ltbl[Lsiz];
struct space Cspace[Csiz];
FILE *fp, *fopen(), *gfile;
int *intrpt();

long
      P,
      CP,
      E,
      B,
      TR,
      H,
      HB,
      S,
      W,
      E_FF = 0,
      cut = 0;
      AX[Rsiz];
long H2;
long cpid = 0;
long parent = -1;
long ictr = 0;
int PROCESSORS = 16;     /* number of processors in system */
short PROCESSOR;         /* current processor being simulated */
float *TIME, *EF, *RDS, *WTS, *CSEC;

#ifdef micro
long PDL = 0;
```

```
#else
long PDL = Dsiz;
#endif

#ifdef micro
long  T,T1,R,
      MAR,MDR,
      MISC;
int   XY,Ui,uRP,nuPC,uPC = -2;
long  arg1,arg2,arg3,arg4;
long  bPDL,PDL1[512], PDLr[512];
#endif

int  N = 0,
     mode, bufval, Wbase, HPbase, STKbase, TRLbase;

long *stack;
struct strng name = { 0, " " };
struct strng f_name;
struct tbl decode_tbl[NUMINSTS];
struct tbl bi_tbl[40];
struct tbl Consts[512];
int quit = 0, dbg = 0, silent = 1, mix = 0, lP, scheck = 0;
short ANDFLAG = 0, ORFLAG = 0, IBFLAG = 0;
extern int inst_total;

main( argc, argv )
   int argc;
   char *argv[];

{  int i, c, spec;
   char *s;
   int ltbl = 0, ptbl = 0, obj = 0;

   for( i = 0; i < Rsiz; i++ ) AX[i] = tag( -1, tvar );
#ifdef micro
   if(( fp = fopen("/b/hprg/tep/plm/level2/notes", "r" )) != NULL )
      while(( c = getc(fp)) != EOF ) putc( c, stdout );
#else
   if(( fp = fopen("/a/hprg/fagin/PPP1/notes", "r" )) != NULL )
      while(( c = getc(fp)) != EOF ) putc( c, stdout );
#endif
   if(( fp = fopen("/a/hprg/fagin/PPP1/dsptbl.d", "r" )) == NULL )
      printf(" file dsptbl unavailable \n" );
   else  {
   lddsptbl( fp, decode_tbl );
   if(( fp = fopen("/a/hprg/fagin/PPP1/bitbl.d", "r" )) != NULL )
      lddsptbl( fp, bi_tbl );
   while((--argc > 0) && ((*++argv)[0] == '-' ))
      for( s = argv[0]+1; *s!='\0'; s++ )
```

```
        switch( *s )
        {  case 'l' : ltbl = 1;
                      break;
           case 'p' : ptbl = 1;
                      break;
           case 'd' : dbg = 1;
                      break;
           case 'c' : obj = 1;
                      break;
           case 'i' : mix = 1;
                      if((gfile = fopen(GRAPH_FILE,"w"))==NULL) {
                         printf(" unable to open graph file\n" );
                         exit(1);
                         }
                      break;
           case 's' : scheck = 1;
                      break;
           case 'n' : PROCESSORS = atoi(s+1);
                      break;
                                    };
     initmix();
     if( argc != 1 )
     {  printf(" Enter program \n");
        f_name = name;     }
     else
     {  sprintf( f_name.ch, " %-1s", *argv );
        f_name.len = 79;   };
     if( load( f_name )) {
        fp = stdin;
        if( silent ) signal( SIGINT, intrpt );
        if( obj  ) Cprn( 0, Caddr );
        if( ptbl ) Pprn( 1, Pno );
        if( ltbl ) Lprn( 1, Lbloc );
        init();
        if( dbg  ) debug( "dbg" );
        else
           while( !quit )
              simulate();
        printf("Stopped.\n");
        getrusage(0,&buffer);
        printf("cpu time is %ld.%06ld sec \n", buffer.ru_utime.tv_sec,
        buffer.ru_utime.tv_usec );
        if( mix && ( inst_total != 0 ) && ( length( name ) != 0 )) {
           prntmix( f_name );
           ggraph(f_name);
           }
        }
     }
}
```

```
/*  file  prns.c  */
/*  seq   1a.2.6   */

#include "regs.h"
#include "parameters.h"
#include "structures.h"
#include "spacevars.h"
#include "typvars.h"
#include "process.h"
#include "osdefs.h"
#include "orp.h"
#include "ib.h"
#include "mem.h"
#include "env.h"
#include "cp.h"
#include "msg.h"

extern long *processors;
extern struct process *pt[];
extern int fl,esc;

Cprn( fm,to )
long fm,to;

{  int i;
if (fm < 0)
fm = P;
if (to < 0)
  to = fm;
   for( i= fm; i <= to; i++ )
   {  printf("CLOC %08x: ", i);
      outputstr( Cspace[i].inst );
      printf("\t");
      outputstr( Cspace[i].args );
#ifdef micro
      if( uPC != -2 ) printf(" :micro%02d", Ui);
#endif
      fl = 0; esc = 0;
      printf("\n");             };  }

Pprn( fm,to )
   long fm,to;

   {  int i;

   if( to > Pno ) to= Pno;
   for( i= fm; i <= to; i++ )
   {  outputstr( Ptbl[i].name );
      printf("\tCLOC %08x\n",Ptbl[i].addr );
```

```
                                              };  }

Lprn( fm,to )
   long fm,to;

{  int i;

   if( to > Lbloc ) to= Lbloc;
   for( i= fm; i <= to; i++ )
   {  outputstr( Ltbl[i].name );
      printf("\tCLOC %08x\n", Ltbl[i].addr );  };  }

PTprn(d) /* print a process table entry */
long d;
{
struct process *p = pt[d];
if (d < 0)
   return;
if (!(pt[d] -> occ)) {
   printf("No such process\n");
   return;
   }
printf("PROCESS %x\n", d);
printf("parent = %08x; kind = %s; c0 = %s; c1 = %08x; state = %s\n",
p -> parent,kindstr(p->kind),comstr(p->c0),p->c1,statestr(p -> state));
printf("AX[1] = %08x;\t %08x;\t %08x;\t %08x;\n",
p -> AX[0],p -> AX[1],p -> AX[2],p -> AX[3] );
printf("AX[5] = %08x;\t %08x;\t %08x;\t %08x;\n",
p -> AX[4],p -> AX[5],p -> AX[6],p -> AX[7] );
printf("P = %08x;\t CP = %08x;\t E = %08x;\t B = %08x\n",
p -> P, p -> CP, p -> E, p -> B);
printf("TR = %08x;\t H = %08x;\t HB = %08x;\t S = %08x\n",
p -> TR, p -> H, p -> HB, p -> S);
printf("N = %08x;  W = %08x;  PDL = %08x;  E_FF = %08x\n",
p -> N, p -> W, p -> PDL, p -> E_FF);
printf("cut = %08x;  ictr = %08x;  parB = %08x; i = %08x\n",
p -> cut, p -> ictr, p -> parB, p -> i);
printf(
"Wbase = %08x;  HPbase = %08x;  STKbase = %08x;  TRLbase = %08x\n",
p -> Wbase, p -> HPbase,p -> STKbase,p -> TRLbase);
}

EXprn(e) /* print an extended environment */
long e;
{
int i;
if (e == -1)
   e = E;
for (i = e; i < e + 9; i++) {
   switch(i-e) {
      case sCE     : printf("E\t");
                     break;
      case sCB     : printf("B\t");
                     break;
      case sCP     : printf("CP\t");
                     break;
      case sCN     : printf("N\t");
                     break;
      case BT_TBL  : printf("BT_TBL\t");
                     break;
      case CP_TBL  : printf("CP_TBL\t");
                     break;
      case LB      : printf("LB\t");
                     break;
      case P_TBL   : printf("P_TBL\t");
                     break;
      case J_TBL   : printf("J_TBL\t");
                     break;
      }
   printf("%08x: %08x\n",i,stack[i]);
   }
}

CPprn(b) /* print a choice point */
long b;
{
int i;
short a = 0;
if (b == -1)
   b = B;
if (cptype(stuck(b-sB)) == orcp) {
   for (i = b-ocp-stuck(b-soL)+1; i < b; i++) {
      switch(b-i) {
         case soB : printf("B\t");
                    break;
         case soN : printf("N\t");
                    break;
         case soE : printf("E\t");
                    break;
         case soCP: printf("CP\t");
                    break;
         case soBP: printf("BP\t");
                    break;
         case soH : printf("H\t");
                    break;
         case soTR: printf("TR\t");
                    break;
         case soL : printf("L\t");
                    break;
         case soC:  printf("CC\t");
```

```
                    break;
          case soW:  printf("W\t");
                     break;
          case soAR: printf("A0\t");
                     break;
          case soAR+1: printf("A1\t");
                       break;
          case soAR+2: printf("A2\t");
                       break;
          case soAR+3: printf("A3\t");
                       break;
          case soAR+4: printf("A4\t");
                       break;
          case soAR+5: printf("A5\t");
                       break;
          case soAR+6: printf("A6\t");
                       break;
          case soAR+7: printf("A7\t");
                       break;
          default    : printf("ch %d\t",b-i-ocp+1);
                       break;
          }
          printf("%08x: %08x\n", i, stack[i]);
       }
    return;
    }

if (cptype(stuck(b-sB)) == pseudocp) {
    for (i = b-3; i < b; i++) {
       switch (b-i) {
          case sB:
             printf("B\t");
             break;
          case sN:
             printf("n\t");
             break;
          case sBCE:
             printf("E\t");
             break;
          }
       printf("%08x: %08x\n", i, stack[i]);
       }
    return;
    }

if ((cptype(stuck(b-sB)) == andcp) || (cptype(stuck(b-sB)) == d_andcp))
    a = 1;
for (i = b-scp; i < b; i++) {
    switch (b-i) {
       case sTR  : printf("TR\t");
```

```
                        break;
            case sH   : printf("H\t");
                        break;
            case sBP  : if (a)
                           printf("M\t");
                        else
                           printf("BP\t");
                        break;
            case 5    : printf("A1\t");
                        break;
            case 6    : printf("A2\t");
                        break;
            case 7    : printf("A3\t");
                        break;
            case 8    : printf("A4\t");
                        break;
            case 9    : printf("A5\t");
                        break;
            case 10   : printf("A6\t");
                        break;
            case 11   : printf("A7\t");
                        break;
            case 12   : printf("A8\t");
                        break;
            case sBCP : printf("CP\t");
                        break;
            case sBCE : printf("E\t");
                        break;
            case sN   : printf("N\t");
                        break;
            case sW   : printf("W\t");
                        break;
            case sB   : printf("B\t");
                        break;
            }
      printf("%08x: %08x\n", i, stack[i]);
      }
}

Eprn(e) /* print an ordinary environment */
long e;
{
int i;
if (e == -1)
   e = E;
for (i = e; i < e + 4; i++) {
   switch(i-e) {
      case sCE    : printf("E\t");
                    break;
      case sCB    : printf("B\t");
```

```
                    break;
      case sCP      : printf("CP\t");
                    break;
      case sCN      : printf("N\t");
                    break;
      }
   printf("%08x: %08x\n",i,stack[i]);
   }
}

Dprn( fm,to )
   long fm,to;
{
long i;
if (fm < 0)
   return;
if (to < 0)
  to = fm;
for ( i = fm; i <= to; i++ )
   printf("DLOC %08x: %08x \n", i, stack[i] );
}

Rprn()

{  char *mstr;
   int x;

   printf("P  = %08x;\t CP = %08x;\t E  = %08x;\t B  = %08x;\n", P,CP,E,B );
   printf("TR = %08x;\t H  = %08x;\t HB = %08x;\t S  = %08x;\n", TR,H,HB,S);
   if( mode == 0 ) mstr = "write";
      else mstr = "read ";
   printf("N  = %08x;\t PDL= %08x;\t W  = %08x;\tmode = %s\n",
   N,PDL,W,mstr);
   if (cut & cutm) x = 1;
      else x = 0;
   printf("cut_flag = %d;\t E_FF = %d;\n", x,E_FF);
   printf("AX[1] = %08x;\t %08x;\t %08x;\t %08x;\n",AX[0],AX[1],AX[2],AX[3] );
   printf("AX[5] = %08x;\t %08x;\t %08x;\t %08x;\n",AX[4],AX[5],AX[6],AX[7] );

#ifdef micro
   printf("\nT  = %08x;\t T1 = %08x;\t R  = %08x;\n",T, T1, R );
   printf("MAR= %08x;\t MDR= %08x;\tMISC= %08x;\n", MAR, MDR, MISC );
#endif

                              }

lstprn( loc )
   int loc;

{
   if( loc < 0 ) printf("No list given \n");
   else
   {  while(( value(stack[loc]) != NIL ) &&
            (!(( stack[loc] & tcdr ) && (type(stack[loc]) == tvar))) &&
            ( stack[loc] != 0 ))
      {  Dprn(loc, loc );
         if( stack[loc] & tcdr ) loc = value(stack[loc]);
            else loc = loc+1;    };
      Dprn( loc, loc );    };  }


Sprn() /* print system status: processors, pids, states */
{
short i;
for (i = 0; i < PROCESSORS; i++) {
   if (processors[i] != -1)
      printf("%x\t%x\t%s\n",i,processors[i],
      statestr(pt[processors[i]]->state));
   }
}

char *kindstr(k)
short k;
{
switch (k) {
   case AND: return("AND");
   case DET_AND: return("DET_AND");
   case OR: return("OR");
   default: return("unknown");
   }
}

char *comstr(c)
long c;
{
switch (c) {
   case FORWARD: return("FORWARD");
   case BACKWARD: return("BACKWARD");
   case FAIL_FROM_AND: return("FAIL_FROM_AND");
   case FAIL_FROM_OR: return("FAIL_FROM_OR");
   case KILL_DESC_AND_DIE: return("KILL_DESC_AND_DIE");
   default: return("unknown");
   }
}

char *statestr(s)
short s;
{
switch (s) {
   case RUNNABLE: return("RUNNABLE");
   case RUNNING: return("RUNNING");
   case SLEEPING: return("SLEEPING");
   }
}

char *msgstr(m)
short m;
{
switch (m) {
   case SUC: return("SUC");
   case FAIL: return("FAIL");
   case NA: return("NA");
   case KILL: return("KILL");
   case CUT: return("CUT");
   default: return("unknown");
   }
}

Wprn(d)
long d;
{
long start,id,i,finish;

if (d == 0) {
   printf("Top level process has no binding window\n");
   return;
   }
if (d == -1) {
   start = Wbase;
   id = cpid;
   finish = W;
   }
else {
   start = pt[d] -> Wbase;
   id = d;
   finish = pt[d] -> W;
   }
printf("Binding window of process %x\n", id);
for (i = start; i < finish; i++) {
   if ((WINSIZE-i) % 2 == 0)
      printf("addr\t ");
   else
      printf("val \t ");
   printf("%08x\t%08x\n", i, stuck(i));
   }
}
```

```
/*  file  putl.c  */
/*  seq   1.3.8   */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "timing.h"

put_variable( arg )
   struct strng arg;

{  int i,n;
   char c;
   long val;

   if( !get_2args( arg, &c, &i, &n )) NOP();
   else if( c == 'Y' ) {
      TP(6);
      val = tag( E+envsz()+n, tvar );
      AX[i] = val;
      stick(E+envsz()+n, val);     }
   else {
      TP(4);
      val = tag( H, tvar );
      AX[i] = val;
      AX[n] = val;
      stick(H++, val);         };   }

put_value( arg )
   struct strng arg;

{  int i,n;
   char c;

   if( !get_2args( arg, &c, &i, &n )) NOP();
   else if( c == 'Y' ) {
      TP(6);
      AX[i] = dereference(stuck(E+envsz()+n));
      }
   else {
      TP(2);
      AX[i] = dereference(AX[n]);
      }
}

put_unsafe_value( arg )
   struct strng arg;
```

```
{  int i,n;
   char c;
   long var,val;
   short flag = 0;

   if(( !get_2args( arg, &c, &i, &n )) || ( c != 'Y' )) NOP();
   else  {
   var = dereference( stuck(E+envsz()+n));
   if( type( var ) == tvar ) {
      if (value(var) > E) {
         if (value(var) < E + envsz() + N) {
            val = tag( H, tvar );
            AX[i] = val;
            stick(H++, val);
            TP(13);
            flag = 1;
            }
         else
            TP(12);
         }
      else
         TP(8);
      }
   else
      TP(6);
   if (!flag)
      AX[i] = var;
   }
}

put_constant( arg )
   struct strng arg;

{  int i,x;
   struct strng C,Ai;

   x = delimit( arg, ',' );
   if( x == 0 ) NOP();
   else  {
   C = substr( arg, 1, x-1 );
   Ai = substr( arg, x+1, length(arg)-x );
   i = get_Areg( Ai );
   if( i < 0 ) NOP();
   else  {
   TP(2);
   AX[i] = get_consts( C );
                                     };    };    }
```

```
/*  file  put2.c  */
/*  seq   1a.3.9   */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "timing.h"


put_nil( arg )
   struct strng arg;

{  int i,x;

   i = get_Areg( arg );
   if( i < 0 ) NOP();  else  {
   TP(2);
   AX[i] = tag( NIL, tcon );    };  }

put_structure( arg )
   struct strng arg;

{  int i,x;
   long val;
   struct strng F,Ai;

   x = delimit( arg, ',' );
   if( x == 0 ) NOP();  else  {
   F = substr( arg, 1, x-1 );
   Ai = substr( arg, x+1, length(arg)-x );
   i = get_Areg( Ai );
   if( i < 0 ) NOP();  else  {
   TP(3);
   val = get_consts( F );
   AX[i] = tag( H, tstr );
   stick(H++, val);
   mode = write;      };  };  }

put_list( arg )
   struct strng arg;

{  int i;

   i = get_Areg( arg );
   if( i < 0 ) NOP();  else  {
   TP(3);
   AX[i] = tag( H, tlst );
   mode = write;             };  }
```

```
/*  file regutils.c  */
/*  seq   1a.4.4      */

#include "structures.h"
#include "spacevars.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"

#define NUMASK 0x07ffffff

extern struct strng BLANK;

get_2args( arg, V, i, n )
   struct strng arg;
   char *V;
   int *i, *n;

{  int x, y=1;
   struct strng Vn, Ai;

   x = delimit( arg, ',' );
   if( x == 0 ) y--;
   else  {
   Vn = substr( arg, 1, x-1 );
   Ai = substr( arg, x+1, length(arg)-x );
   *i = get_Areg( Ai );
   *n = get_XYreg( Vn );
   if( *n < 0 ) *n = get_Areg( Vn );
   if(( *i < 0 ) || ( *n < 0 )) y--;
      else *V = getch( Vn, 1 );     };
   return( y );                     }

get_Areg( A )
   struct strng A;

{  int x;

   if((( x = delimit( A, 'A' )) == 0 ) &&
       (( x = delimit( A, 'X' )) == 0 )) return( -1 );
   else return( getnum( substr( A, x+1, length(A)-x))-1);  }

get_XYreg( V )
   struct strng V;

{  int x;

   if((( x = delimit( V, 'X' )) == 0 ) && (( x = delimit( V, 'Y' )) == 0 ))
      return( get_Areg( V ) );
   else return( getnum( substr( V, x+1, length(V)-x)) -1); }

long get_consts( C )
   struct strng C;

{  long x;

   if( getch( C, 1 ) == '&' )
   {  C = substr( C, 2, length(C)-1 );
      x = ( getnum( C ) & NUMASK ) | ( tnum << 27 );  }
   else if(( getch( C, 1 ) >= '0' ) && ( getch( C, 1 ) <= '9' ))
      x = ( getnum( C ) & NUMASK ) | ( tnum << 27 );
   else  {
   x = locate( Consts, Cno, C );
   if( x < 0 )
   {  Consts[++Cno].name = C;
      Consts[Cno].addr = Caddr;
      Cspace[Caddr].inst = C;
      Cspace[Caddr++].args = BLANK;
      x = Consts[Cno].addr;                 }
  else x - Consts[x].addr;                   };
   return( tag( x, tcon) );                  }

struct strng makenum( val )
   long val;

{  struct strng num;
   int i;

   sprintf( num.ch, " %-11d", val );
   for( i = 0; num.ch[i] != '\0'; i++ );
   num.len = i - 1;
   return( num );                   }
```

```
#include <stdio.h>
#include "osdefs.h"
#include "regs.h"
#include "process.h"
#include "parameters.h"
#include "spacevars.h"
#include "instvars.h"
#include "graph.h"

extern int quit, dbg, BKPT, BKPROC;
extern long ictr;
extern struct process *pt[];

extern int esc, mix;
long *processors;
int sim_cyc = 1;
float avg = 0;
extern FILE *gfile;

simulate()
{
short i,j,k;
long dummy[MAXPROCS];
for (PROCESSOR = 0; PROCESSOR < PROCESSORS; PROCESSOR++) {
      sim();
      if (quit)
         return;
      }
j = running_processes();
k = runnable_processes(dummy);
if ((j == 0) && (k == 0)) {
   printf("DEADLOCK!!\n");
   debug("dbg");
   }
if (mix)
   fprintf(gfile,"%d    %d\n", sim_cyc, k+j);
avg = (avg*(sim_cyc-1)+k+j)/sim_cyc;
sim_cyc++;
}

int bkflag = 0;
extern char *comstr();
sim()
{
long pid;
if (processors[PROCESSOR] == -1)
   return;
pid = processors[PROCESSOR];
if (pt[pid] -> state != RUNNING)
   return;
```

```
config_sim(pid);
if (BKPT == P)
   if ((BKPROC == -1) || (BKPROC == cpid))
      bkflag = 1;
if (tracing(cpid))
   bkflag = 1;
if ((DBT) || (bkflag)) {
   printf("Running process %x ", cpid);
   if (c0 == FORWARD) {
      printf("\n");
      Cprn(P,P);
      }
   else
      printf("%s\n", comstr(c0));
   }
dispatch();
/*
if ((DBT) || (bkflag)) Rprn();
*/
if (bkflag) printf("BREAKPOINT\n");
save(pid);
}

config_sim(pid)
long pid;
{
short i;
struct process *p;
p = pt[pid];
for (i = 0; i < Rsiz; i++)
   AX[i] = p -> AX[i];
E = p -> E;
B = p -> B;
N = p -> N;
H = p -> H;
P = p -> P;
CP = p -> CP;
TR = p -> TR;
HB = p -> HB;
S = p -> S;
W = p -> W;
E_FF = p -> E_FF;
cut = p -> cut;
mode = p -> mode;
parent = p -> parent;
cpid = p -> pid;
ictr = p -> ictr;
Wbase = p -> Wbase;
HPbase = p -> HPbase;
STKbase = p -> STKbase;
```

```
TRLbase = p -> TRLbase;
PDL = p -> PDL;
c0 = p -> c0;
c1 = p -> c1;
kind = p -> kind;
}

save(pid)
long pid;
{
short i;
struct process *p;
p = pt[pid];
for (i = 0; i < Rn; i++)
   p -> AX[i] = AX[i];
p -> E = E;
p -> B = B;
p -> N = N;
p -> H = H;
p -> P = P;
p -> CP = CP;
p -> TR = TR;
p -> HB = HB;
p -> S = S;
p -> W = W;
p -> mode = mode;
p -> E_FF = E_FF;
p -> cut = cut;
p -> ictr = ictr;
p -> c0 = c0;
p -> c1 = c1;
}
```

```
/*  file  specials.c  */
/*  seq   1.3.15      */

#include "regs.h"
#include "structures.h"
#include "spacevars.h"
#include "typvars.h"
#include "routines.h"
#include "parameters.h"
#include "instvars.h"

extern int quit;
extern int lP;
extern int dbg;
int mrk;

pquit()

{
   quit = 1;
}

NOP()

{
#ifdef micro
   prefetch(1);
#endif
   printf(" in NOP : " );
   Cprn( lP, lP );      }

mark( arg )
   struct strng arg;

{
#ifdef micro
   prefetch(1);
#endif
   mrk = 1;
}

smode()
{
if (mode == read)
   mode = write;
else if (mode == write)
   mode = read;
}
```

```
/*  file  strngio.c  */
/*  seq   1.4.1      */

#include <stdio.h>
#include "structures.h"

extern struct strng endstr;

struct strng inputstr( fp )
   FILE *fp;

{  int i;
   char ch;
   struct strng line;

   i = 1;
   line.ch[0] = ' ';
   fscanf( fp, "%c", &ch );
   if( ch == EOF ) return( endstr );
   while( ch != '\n' )
   {  if( ch == '\t' ) ch = ' ';
      if(( ch != ' ' ) || (( ch == ' ' ) && ( line.ch[i-1] != ' ')))
      {  line.ch[i] = ch; i++;  };
      fscanf( fp, "%c", &ch );   };
   line.ch[i] = '\0';
   line.len = i - 1;
   return( line );
                                            }

outputstr( line )
   struct strng line;

{  int i;

   if( line.len > 0)
   {  for( i = 1; i <= line.len; i++)
         printf("%c", line.ch[i] ); };    }


getnum( line )
   struct strng line;

{  int i;

   line.ch[line.len+1] = '\0';
   sscanf( line.ch, "%d", &i );
   return( i );                  }
```

```
long gethex( line )
   struct strng line;

{  long i;

   line.ch[line.len+1] = '\0';
   sscanf( line.ch, "%lx", &i );
   return( i );                  }


char getch( line, index )
   struct strng line;
   int index;

{   return( line.ch[index] );    }
```

```
/*  file  strngutils.c  */
/*  seq   1.4.0         */

#include "structures.h"

delimit( line, ch )
   struct strng line;
   char ch;

{  int i;

   for( i = 1; (( i <= line.len ) && ( line.ch[i] != ch ) &&
         ( line.ch[i] != '\0' )); i++ );
   if( i > line.len ) i = 0;
   return( i );
                                          }

length( line )
   struct strng line;

{
   return( line.len );                    }


struct strng substr( line, start, lenth )
   struct strng line;
   int start,lenth;

{  struct strng nline;
   int limit, j;

   nline.ch[0] = ' ';
   nline.len = lenth;
   if( start + lenth - 1 > length( line ) ) limit = length( line ) - start;
   else limit = lenth;
   for( j = 1; j <= limit; j++ )
      nline.ch[j] = line.ch[start++];
   for( j = limit; j < lenth; j++ )
      nline.ch[j] = ' ';
   nline.ch[lenth+1] = '\0';
   return( nline );
                                                       }

equalstr( l1, l2 )
   struct strng l1,l2;

{  int i,result;

   if( length( l1 ) != length( l2 )) result = 0;
   else
   {  result = 1;
      for( i = 0; (( i <= length( l1 )) && result ); i++ )
         if( l1.ch[i] != l2.ch[i] ) result = 0;             }
   return( result );
                                                             }

struct strng concat( l1, l2 )
   struct strng l1,l2;

{  struct strng l3;
   int i,temp;

   temp = length(l1);
   for( i = 0; i <= length( l1 ); i++ )
      l3.ch[i] = l1.ch[i];
   for( i = 1; i <= length( l2 ); i++ )
      l3.ch[temp+i] = l2.ch[i];
   l3.len = temp + length(l2);
   return( l3 );   }
```

```
/*  file  tblutils.c  */
/*  seq   1.4.2       */

#include <stdio.h>
#include "structures.h"
#include "spacevars.h"
#include "instvars.h"

locate( itbl, size, S )
   struct tbl itbl[];
   struct strng S;
   int size;

{  int i = 0, found = 0;

   while(( i <= size ) && ( found == 0 )) {
      if( equalstr( S, itbl[i].name )) found = 1;
      else i++;
      }
   if( found == 0 ) return( -1 );
   else return( i );                             }

lddsptbl( fp, dtbl )
   FILE *fp;
   struct tbl dtbl[];

{  int i;
   struct strng line;
   struct strng inputstr();

   for( i = 0; i < NUMINSTS; i++ )
   {  line = inputstr( fp );
      dtbl[i].name = line;    };    }
```

```
/*  file  unify0.c  */
/*  seq   1a.3.11   */

#include "regs.h"
#include "structures.h"
#include "typvars.h"
#include "parameters.h"
#include "routines.h"
#include "instvars.h"
#include "timing.h"

extern int mixtbl[];

unify( arg1, arg2 )
   long arg1, arg2;

{  long temp;

   mixtbl[UNIFS]++;  mixtbl[UNIFR]++;
   if(( type( arg1 ) == tvar ) || ( type ( arg2 ) == tvar )) {
      if(( type ( arg1 ) == tvar ) && ( type(arg2) == tvar))
         if(value(arg1) < value(arg2))
            bind( arg1, arg2 );
         else
            bind( arg2, arg1 );
       else if( type(arg1) == tvar)
          bind( arg2, arg1 );
       else
          bind( arg1, arg2 );
       TP(2);
       return( 1 );
       }
   else if(( type(arg1) == tcon ) && ( arg1 == arg2)) {
      TP(3);
      return( 1 );
      }
   else if((( type(arg1) == tlst ) && ( type(arg2) == tlst )) ||
           (( type(arg1) == tstr ) && ( type(arg2) == tstr )))
   {  mixtbl[UNIFS]--;
      if( !unify( stuck(value(arg1)), stuck(value(arg2)) )) {
         TP(2);
         return(0);
         }
      else {
         TP(8);
         temp = value(arg1)+1;
         arg1 = stuck( temp );
         if(!( arg1 & tcdr )) arg1 = tag( temp, tlst );
         temp = value(arg2)+1;
```

```
            arg2 = stuck( temp );
            if(!( arg2 & tcdr )) arg2 = tag( temp, tlst );
            if( !unify( arg1, arg2 ))
                return(0);
            else return(1);     };                                  }
    else {
        TP(2);
        return( 0 );
        }
}
```

```
/*  file  unify1.c  */
/*  seq   1a.3.12    */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "timing.h"

unify_void( arg )
   struct strng arg;

{  int i,n;
   long val;

   if( length( arg ) == 0 ) NOP();
   else  {
   n = getnum( arg );
   TP(1);
   switch( mode )
   {  case read   : for( i = 0; i < n; i++ ) {
                       val = decdr( stuck( S ) );
                       if(( val & tcdr ) && ( type(val) != tvar )) {
                          TP(6);
                          fail();
                          break;
                          }
                       else if( !( val & tcdr ) )
                          TP(4);
                       else {
                          TP(8);
                          mode = write;
                          n = n - i;
                          stick( value(val), tag( H, tlst ) | tcdr );
                          break;
                          }
                    }
                    if( mode == read ) break;
      case write  : TP(2);
                    for( i = 0; i < n; i++ ) {
                       stick(H++, tag( H, tvar ));
                       TP(2);
                       }
                                                   };   };  }

unify_variable( arg )
   struct strng arg;

{  int n,typ;
```

```
   char c;
   long val;

   if(( n = get_XYreg( arg )) < 0 ) NOP();  else  {
   c = getch( arg, 1 );
   switch ( mode )
   {  case read  : val = decdr( stuck( S ) );
                   if(( val & tcdr ) && ( type(val) != tvar )) {
                      TP(4);
                      fail();
                      }
                   else if(( val & tcdr ) && ( type(val) == tvar )) {
                      TP(7);
                      unify( val, tag( H, tlst ) | tcdr );
                      mode = write;  }
                   else {
                      TP(2);
                      break;
                      }
      case write : val = tag( H, tvar );
                   TP(5);
                   stick(H++, val);   };
   if( c == 'Y' ) {
      TP(2);
      stick(E+envsz()+n, val);
      }
   else  AX[n] = val;                   };  }

unify_value( arg )
  struct strng arg;

{  int n;
   char c;
   long var,val;

   if(( n = get_XYreg( arg )) < 0 ) NOP();  else  {
   c = getch( arg, 1 );
   if( c == 'Y' ) {
      var = dereference( stuck(E+envsz()+n) );
      TP(6);
      }
   else {
      var = dereference( AX[n] );
      TP(2);
      }
   switch( mode )
   {  case read  : val = decdr( stuck( S ) );
                   if(( val & tcdr ) && ( type(val) != tvar )) {
                      TP(5);
                      fail();
```

```
                                  }
                               else if(( val & tcdr ) && ( type(val) == tvar )) {
                                  TP(7);
                                  unify( val, tag( H, tlst ) | tcdr );
                                  mode = write;  }
                               else {
                                  TP(2);
                                  val = dereference( val );
                                  if( !unify( var, val )) fail();
                                  break; };
              case write :
                               TP(1);
                               stick(H++, var);
                                                                           };  };  }
```

```
/*  file  unify2.c  */
/*  seq   1a.3.13    */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "timing.h"

static struct strng NILstr = { 3, " nil" };

unify_constant( arg )
   struct strng arg;

{  int x;
   long val,var;

   val = get_consts( arg );
   switch( mode )
   {  case read  : var = decdr( stuck( S ) );
                   if(( var & tcdr ) && ( type(var) != tvar )) {
                      TP(4);
                      fail();
                      }
                   else if(( var & tcdr ) && ( type(var) == tvar )) {
                      TP(8);
                      unify( var, tag( H, tlst ) | tcdr );
                      mode = write;
                      }
                   else {
                      TP(5);
                      var = dereference( var );
                      if( !unify( var, val )) fail();
                      break;
                      }
      case write : TP(2);
                   stick(H++, val);
                                                              };  }

unify_nil()

{  long val;

   switch( mode )
   {  case read  : val = stuck(S);
                   TP(5);
                   if(!( val & tcdr ) ||
                      !( unify(val, tag(NIL,tcon)|tcdr))) fail();
                   break;
```

```
        case write :
                        TP(2);
                        stick(H++, tag( NIL, tcon ) | tcdr );   };  }


unify_cdr( arg )
   struct strng arg;

{  int n,typ;
   char c;
   long val;

   if(( n = get_XYreg( arg )) < 0 ) NOP();  else  {
   c = getch( arg, 1 );
   switch ( mode )
   {  case read  : val =  stuck( S );
                   TP(5);
                   if(!( val & tcdr )) val = tag( S, tlst );
                   else val = val ^ tcdr;
                   break;
      case write : val = tag( H, tvar );
                   TP(6);
                   stick(H++, val | tcdr);  };
   if( c == 'Y' ) {
      stick(E+envsz()+n, val);
      TP(2);
      }
   else
      AX[n] = val;
   }
}
```

```
/*  file  unify3.c  */
/*  seq   1a.3.16    */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "parameters.h"
#include "timing.h"

unify_local_value(args)
struct strng args;
{
unify_value(args);
}

unify_local_val( arg )
   struct strng arg;

{  int n;
   char c;
   long var,val;

   if(( n = get_XYreg( arg )) < 0 ) NOP();  else  {
   c = getch( arg, 1 );
   if( c == 'Y' )  var = dereference( stuck(E+envsz()+n) );
   else  var = dereference( AX[n] );
   switch( mode )
   {  case read  : val = decdr( stuck( S ) );
                   if(( val & tcdr ) && ( type(val) != tvar )) fail();
                   else if(( val & tcdr ) && ( type(val) == tvar ))
                   {  stick( value(val), tag( H, tlst ) | tcdr );
                      mode = write;  }
                   else
                   {  val = dereference( stuck(S++) );
                      if( !unify( var, val )) fail();
                      break; };
      case write : if(( type( var ) == tvar ) && ( value( var ) > STKbase ))
                   {  stick( H-1, tag( H+1, tlst) | tcdr );
                      var = tag( H, tvar );
                      stick( H++, var );
                      if( c == 'X' ) AX[n] = var;  };
                   stick(H++, var);
                                                     };  };  }
```

```
/*  file  unused.c   */
/*  seq   1.3.16     */

#include "structures.h"
#include "regs.h"
#include "typvars.h"
#include "routines.h"
#include "parameters.h"

unify_local_value( arg )
   struct strng arg;

{  int n;
   char c;
   long var,val;

   if(( n = get_XYreg( arg )) < 0 ) NOP();  else  {
   c = getch( arg, 1 );
   if( c == 'Y' )  var = dereference( stuck(E+env+n) );
   else  var = dereference( AX[n] );
   switch( mode )
   {  case read  : val = dereference( stuck(S++) );
                   if( !unify( var, val )) fail();
                   else if( c =='X' ) AX[n] = val;
                   S = value( stuck(S) );
                   break;
      case write : if(( type( var ) == tvar ) && ( value( var ) > STKbase ))
                   {  stick( H-1, tag( H+1, tlst ));
                      var = tag( H, tvar );
                      stick( H++, var );
                      if( c == 'X' ) AX[n] = var;  };
                   stick(H++, var);
                   stick(H++, tag( H+1, tlst ));
                                                    };  };  }
```

```
#define sW (Rn+8)
#define sTR (Rn+7)
#define sH (Rn+6)
#define sBP (Rn+5)       /* if SEQ choice point */
#define sM (Rn+5)        /* if AND choice point */
#define sBCP 4
#define sBCE 3
#define sN 2
#define sB 1

#define scp (Rn+8)       /* SEQ choice point size */
#define cutm 0x20000000
#define seqcp 0x00000000
#define andcp 0x40000000
#define d_andcp 0x60000000
#define ht 0x80000000
#define soll 0x40000000

#define ID(x) (x & 0x1fffffff)
#define SAT (running_processes() == PROCESSORS)
#define SEQFLAG 0x80000000
#define SEQ(x) (x & SEQFLAG)
extern int scheck;
```

```
#define sCE 0
#define sCB 1
#define sCP 2
#define sCN 3
#define env 4
```

```
struct fqentry {
   long par,     /* parent process id */
   kind,         /* AND or OR */
   P,            /* program counter */
   parB,         /* choice point that created process */
   pid,          /* ID of created process */
   i             /* "index": join table entry for AND processes,
                    child # for OR processes */
   };
```

```
#define GRAPH_FILE "/usr/tmp/ppp_graph"
#define MAXPOINTS 500
#define XTICKS 10
#define XSTART 0
#define XORG 170
#define YTICKS 10
#define YSTART 0
#define YORG 15
```

```
#define BT_TBL 4
#define CP_TBL 5
#define LB 6
#define P_TBL 7
#define J_TBL 8
#define envp 9          /* to include TBL pointers */

#define pcp 3           /* PSEUDO choice point size */
#define EXIT 0x80000000
#define pseudocp 0xc0000000
#define UNKNOWN 0x10000000
#define NONE 0x80000000
```

```
/*  file instvars.h  */

/*  seq   1.1.6      */

#define NUMINSTS 61

#define FAILS NUMINSTS
#define TYPEI NUMINSTS+1
#define TYPEII NUMINSTS+2
#define UNIFS NUMINSTS+3
#define UNIFR NUMINSTS+4
#define BINDS NUMINSTS+5
#define ESCPS NUMINSTS+6
#define READS NUMINSTS+7
#define WRITS NUMINSTS+8
#define DERFS NUMINSTS+9
#define TRALS NUMINSTS+10
#define MXTRL NUMINSTS+11
#define MXSTK NUMINSTS+12
#define MXHEP NUMINSTS+13
#define MXPDL NUMINSTS+14
#define MXWIN NUMINSTS+15
#define SWAPS NUMINSTS+16

#define MIXSZ NUMINSTS+17
```

```
#define WINSIZE 64
#define HPSIZE (1024-WINSIZE)
#define STKSIZE 1024
#define TRPDLSIZE 128
#define PMSIZE (WINSIZE + HPSIZE + STKSIZE + TRPDLSIZE)
```

```
#define FAIL 0
#define SUC 1
#define NA 2
#define KILL 3
#define CUT 4
```

```
#define ocp 18  /* or choice point size */

#define soB 1   /* MUST be same as sB!! */
#define soN 2
#define soE 3
#define soCP 4
#define soBP 5
#define soH 6
#define soTR 7
#define soW 8
#define soL 9
#define soC 10
#define soAR 11

#define orcp 0x80000000
#define SUCFLAG 0x40000000
```

```
#define MAXPROCS 512

#define MAXINSTS 16
#define MAXFORKS 16
#define MAXMSGS 16
#define MAXREQS 16

#define TIMEOUT 0
#define SLEEP 1
#define DIE 2

#define FORWARD 0
#define KILL_DESC_AND_DIE 55
#define FAIL_FROM_OR 56
#define FAIL_FROM_AND 57
#define BACKWARD 58

#define RUNNING 1
#define RUNNABLE 2
#define SLEEPING 3

#define DBT (dbg || tracing(cpid))
#define TPROCS 8

extern char *kindstr(), *msgstr(), *statestr(), *comstr();
```

```
/*  file  parameters.h  */
/*  seq   1.1.0         */

#define Psiz 128
#define Isiz 1024
#define Csiz 2048
#define Dsiz (1024*1024)

/* memory allocation of top level process */
#define THPbase 128
#define TSTKbase (16*1024)
#define TTRLbase (31*1024)
#define TPSIZE (32*1024)

extern long Wbase;
extern long HPbase;
extern long STKbase;
extern long TRLbase;
#define Rsiz 8
```

Aug 4 17:31 1986 process.h Page 1

```
#define Rsiz 8
struct process {
   short occ, kind, state;
   long AX[Rsiz],
   P,
   CP,
   E,
   B,
   TR,
   H,
   HB,
   S,
   N,
   W,
   mode,
   E_FF,
   cut,
   ictr,
   parent,
   Wbase,
   HPbase,
   STKbase,
   TRLbase,
   PDL,
   parB,
   i,
   pid,
   c0,c1
   };
```

```
/*  file  regs.h  */
/*  seq   1.1.1   */

#define AND 0
#define OR 1
#define DET_AND 2

#define Rn 8
extern long
      P,
      CP,
      E,
      B,
      TR,
      H,
      HB,
      S,
      W,
      PDL,
      cut,
      E_FF,
      AX[],
      c0,c1,
      cpid,
      parent;
extern int
        N,
        mode;
extern short
        kind;

extern int dbg, PROCESSORS, PROCESSOR;
extern long *stack;
extern struct tbl Consts[];
extern struct strng BLANK;
```

```
struct rqentry {
   short request;
   long pid;
   };
```

```
/*  file  routines.h  */
/*  seq   1.1.5       */

struct strng substr();
struct strng makenum();
struct strng concat();
long value();
long numvalue();
long tag();
long dereference();
long stuck();
long cptype();
long OS_FORK();
long jte();
long gethex();
long emask();
```

```
/*  file  spacevars.h  */
/*  seq   1.1.2        */

extern long  Caddr, Pno, Lbloc, Cno;
extern struct tbl Ptbl[], Ltbl[];
extern struct space Cspace[];
extern int mixtbl[], inst_total;
```

```
/*  file  structures.h  */
/*  seq   1.1.3         */

struct strng {
              int  len;
              char ch[80]; };

struct tbl  {
              struct strng name;
              long   addr;          };

struct space {
               struct strng inst;
               struct strng args;   };
```

```
#define TP(x)  (TIME[PROCESSOR] += (x))
#define GET_LOCK CSEC[PROCESSOR]++
#define RELEASE_LOCK

#define PROC_STATE_DUMP_TIME 25
#define PROC_STATE_LOAD_TIME 25
#define FORK_TIME 10
#define MSG_WRITE_TIME 4


extern float *TIME, *EF, *RDS, *WTS, *CSEC;
```

```
/*  file  typvars.h  */
/*  seq   1a.1.4     */

#define tcdr 0x20000000
#define tlst 0
#define tstr 1
#define tcon 3
#define tvar 2
#define tnum 1

#define write 0
#define read  1
#define NIL -1
```